Physical adsorption analysis

Use this template as a starting point to carry out the analysis tasks. For reference, here are links to recommended Python resources: the Whirlwind Tour of Python and the Python Data Science Handbook both by Jake VanderPlas.

Standard Packages

This is a good idea at the beginning of your notebook to include the packages that you will need. We will use those shown below here. A brief description:

- `numpy` is the foundational package for Python numerical work. It extends and speeds up array operations beyond standard Python, and it includes almost all math functions that you would need for example `sqrt()` (square root) or `cos()` (cosine). These would be written in code as `np.sqrt()` or `np.cos()`.
- `scipy` is a huge collection of scientific data analysis functions, routines, physicical constants, etc. This is the second most used package for scientific work. Here we will use the physical constants library, `scipy.constants`. Documentation is at SciPy.org with the constants subpackage at https://docs.scipy.org/doc/scipy/reference/constants.html.
- `uncertainties` is a very useful small package that simplifies uncertainty propagation and printing out of quantities with uncertainty. Documentation is at https://pythonhosted.org/uncertainties/
- `matplotlib` is *the* standard plotting package for scientific Python. We will use a subset called `pyplot` which is modeled after the plotting functions used in MATLAB. The last line below, `%matplotlib inline`, simply forces the plots to appear within the notebook.
- `pandas` is a large data science package. It's main feature is a set of methods to create and manipulate a "DataFrame," which is an enlargement of the idea of an array. I plays well with NumPy and other packages. We will use it mainly as a way to read files into data sets in an easy way.
- LMFit is excellent for carrying out line and curve fits with many useful features.

Getting Help

See the example code for a wide range of actions in notebooks created by Prof. Marjorie Olmstead and Prof. David Pengra in this repository: **Physics431/Examples**.

You can pull the examples into your environment with the following command. (Only do this once, or you will get an error):

```
git clone https://github.com/Physics431/Examples
```

Task Summary

1. Create a Python code block that will calculate an expected dosing pressure for a next dose, given the current equilibrium pressure, current adsorbed amount, and expected additional adsorbed amount. Use this code to estimate the dosing pressure for the first 4 data points and check with the instructor or TA to make sure you are ready to start taking data.
2. With your software code, set up the ability to make a plot similar to Figure 5 that will allow you to watch the progress of the isotherm as it is created.
3. Collect and plot data for the argon isotherm, per the instructions. Include a table of your pressure measurements before and after each dose along with the calculated adsorbed amount.
4. Repeat the procedure to collect and plot data for nitrogen isotherm. Be careful: the substep will appear between 7 and 9 torr; decrease dose sizes significantly so that you do not miss this feature.
5. Determine the monolayer completion volume-STP of the nitrogen floating phase and the argon by using the “point B” method. You may fit a line to a subset of points in your data, and show this line on the full data set. Also show the location of “point B” for each set.

In [1]:
```
# Usually import packages via a handle to the functions in them using import
#
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
import uncertainties as unc
%matplotlib inline
```

In [2]:
```
# Useful plot default
mpl.rcParams['figure.figsize'] = 12.0,8.0  # Roughly 12 cm wde by 8 cm high
mpl.rcParams['font.size'] = 14.0 # Use 14 point font
```

Calculate expected dosing

Rearrange the formulas giving the coverage amount for each step summarized by the Number versus Pressure calculation to make a formula that predicts the dosing pressure given the sample and calibration volumes $v_s$ and $v_c$, the room temperature $T$ (K), the standard temperature and pressure $T_0 = 273$ K and $P_0 = 760$ torr, and the amount adsorbed in a given step in $V_{STP}$ (in cc).

Then make a Python function that calculates this given the input quantities.

In [3]:
```
### GLOBAL
P_std = 760
T_std = 273
T_rm = 295
v_s = 34.8
v_c = 85.1
R = 62363.577   # units: cc*torr/(K*mol)
NA = 6.022*10**(23)
kB = R/NA

# Create your function here
def dose_pressure(V_ads, P_eq):
    P_c=np.zeros(len(V_ads)-1)

    for i in range(len(P_c)):
        delta_v_ads=V_ads[i+1]-V_ads[i]
        P_s_0=P_eq[i]
        P_s_f=P_eq[i+1]
        P_c[i]=delta_v_ads*(1/v_c)*(P_std*T_rm/T_std)  +  (v_s/v_c)*(P_s_f-P

    return P_c
```

Read the graph showing example data (Figure 5) and extract the adsorbed amount $V_{\rm ads}$ versus pressure $P$ for the first 4 or 5 data points. Feed this data to your function and make a table showing the dosing pressure expected to create those data.

In [4]:
```
# Create arrays to hold the extracted results
V_ads = np.array([0, 2.5, 3.7, 5.1, 6.8, 8.5, 9.8, 10.7, 11.2, 11.3, 11.5, 1
P_eq = np.array([0, 0.3, 0.5, 0.55, 0.6, 0.7, 0.95, 1.9, 4.25, 5.6, 7.8, 9.4

# Feed the arrays to your function above, and print a table
P_c = dose_pressure(V_ads, P_eq)

ArInstructionData = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium
print('Ar instruction data:')
print(ArInstructionData)
```

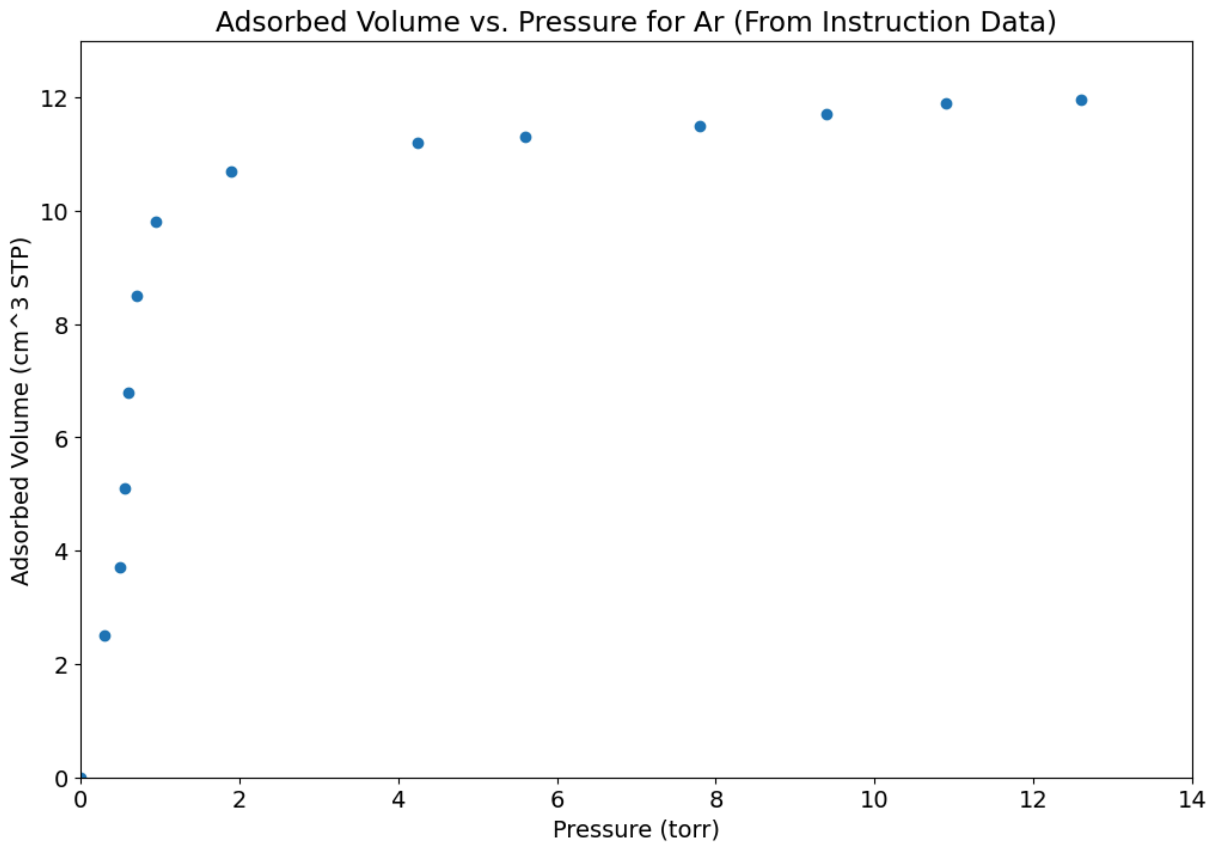
```
Ar instruction data:
    V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0                  0.00                         0.00             24.548573
1                  2.50                         0.30             12.162215
2                  3.70                         0.50             14.080947
3                  5.10                         0.55             17.026054
4                  6.80                         0.60             17.146501
5                  8.50                         0.70             13.597697
6                  9.80                         0.95             10.973806
7                 10.70                         1.90             10.036166
8                 11.20                         4.25              7.117092
9                 11.30                         5.60             10.629719
10                11.50                         7.80             11.984361
11                11.70                         9.40             13.443467
12                11.90                        10.90             13.777700
13                11.95                        12.60              0.000000
```

Check with the instructor or TA to make sure your calculation is correct before starting to dosing procedure.

Set up data plotting

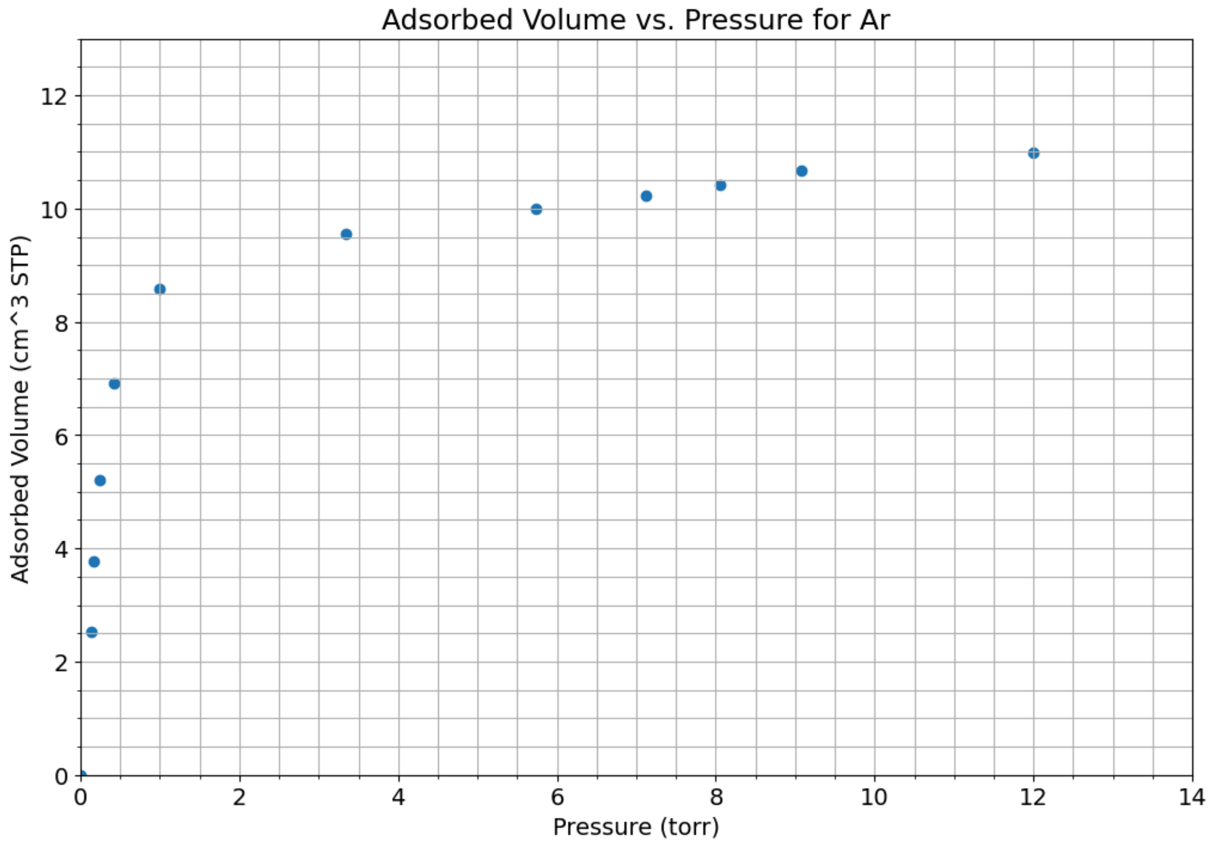
Create another function that will use your initial and final pressure for each point, along with the system parameters (volumes, room temperature) to plot a graph of your data similar to Figure 5 as you collect the data.

In [5]:
```
# Create your function here
def calc_v_ads(P_eq, P_c):
    V_ads = np.zeros(len(P_eq))

    for i in range(len(P_c)):
        delta_P_s = P_eq[i+1]-P_eq[i]
        delta_P_c = P_eq[i+1]-P_c[i]
        delta_v_ads = -(T_std/(P_std*T_rm)) * (v_s*delta_P_s + v_c*delta_P_c
        V_ads[i+1] = V_ads[i] + delta_v_ads

    return V_ads

V_ads = calc_v_ads(P_eq, P_c)

plt.scatter(P_eq, V_ads)
plt.title('Adsorbed Volume vs. Pressure for Ar (From Instruction Data)')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')
plt.xlim(0, 14)
plt.ylim(0, 13)
plt.show()
```



In [6]:
```
# Ar data
P_c = np.array([24.57, 12.19, 14.13, 17.06, 17.23, 13.61, 10.99, 10.04, 10.2
P_eq = np.array([0, 0.14, 0.17, 0.24, 0.42, 0.99, 3.34, 5.74, 7.12, 8.05, 9.

# calculate adsorbed volume
V_ads = calc_v_ads(P_eq, P_c)

# store results in dataframe
Ar_results = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressu
print("Ar Results:")
print(Ar_results)

# plot results
plt.scatter(P_eq, V_ads)
plt.title('Adsorbed Volume vs. Pressure for Ar')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')
plt.grid(which='both')
plt.minorticks_on()
plt.xlim(0, 14)
plt.ylim(0, 13)
plt.show()
```

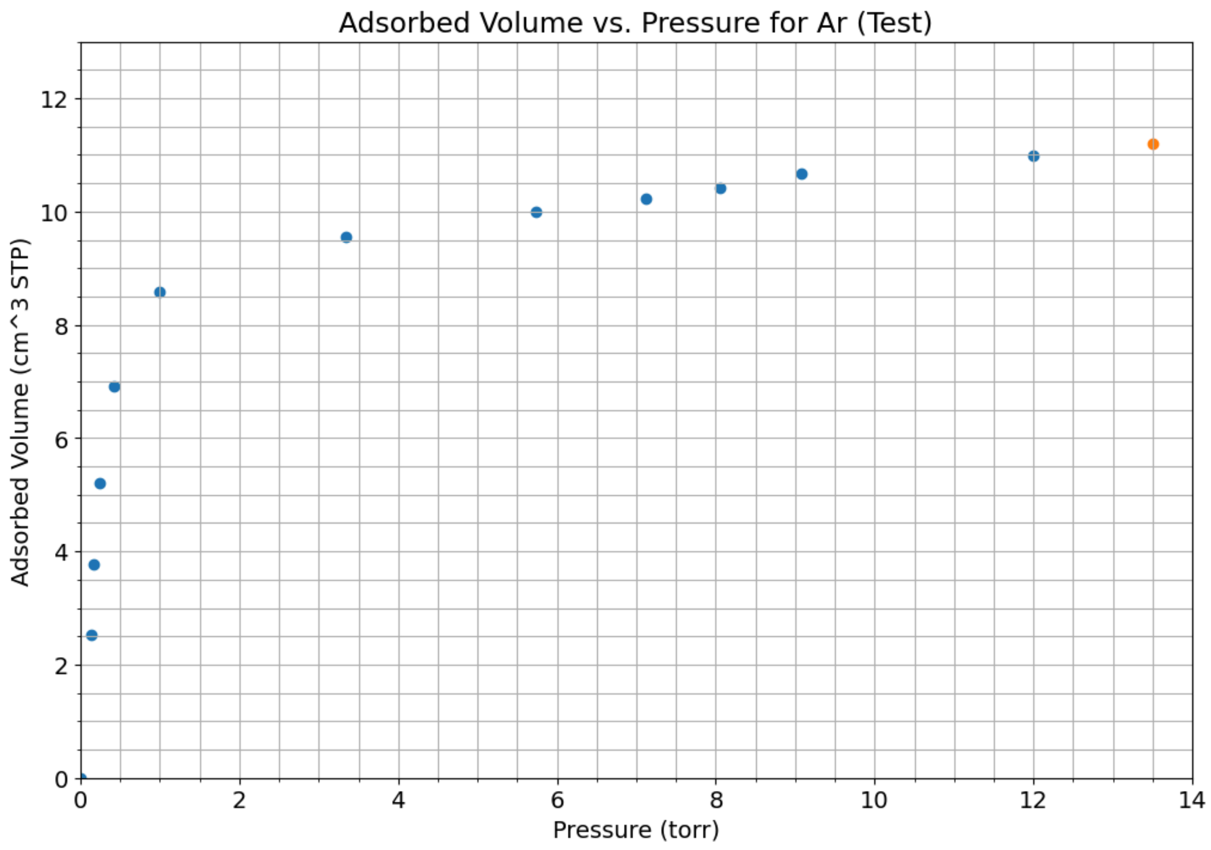
```
Ar Results:
    V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0              0.000000                         0.00                 24.57
1              2.525580                         0.14                 12.19
2              3.769858                         0.17                 14.13
3              5.206217                         0.24                 17.06
4              6.922878                         0.42                 17.23
5              8.581564                         0.99                 13.61
6              9.546193                         3.34                 10.99
7              9.988515                         5.74                 10.04
8             10.232617                         7.12                 10.27
9             10.423252                         8.05                 11.82
10            10.663533                         9.08                 16.37
11            10.994092                        11.99                  0.00
```



In [7]:
```
### This cell is for using existing data to calculate the next dose pressure
### based on the data we took, once the instruction data stops working.

# Existing data:
P_c = Ar_results['Dose Pressure (torr)'][0:-1].to_numpy()
P_eq = Ar_results['Equilibrium Pressure (torr)'].to_numpy()

# Calculate adsorbed STP volume
V_ads = calc_v_ads(P_eq, P_c)

# Estimate the next point and append to your data arrays
P_eq_next=13.5
V_ads_next=11.2

P_eq=np.append(P_eq, P_eq_next)
V_ads=np.append(V_ads, V_ads_next)
```

```
# Feed the arrays to the dose pressure function, and use the dose pressure t
# then feed data back into the previous cell
P_c = dose_pressure(V_ads, P_eq)
print("Estimated Dose pressure:", P_c[-1])

# Plot for visualization
plt.scatter(P_eq[0:-1], V_ads[:-1])
plt.scatter(P_eq[-1], V_ads[-1])
plt.title('Adsorbed Volume vs. Pressure for Ar (Test)')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')
plt.grid(which='both')
plt.minorticks_on()
plt.xlim(0, 14)
plt.ylim(0, 13)
plt.show()
```

```
Estimated Dose pressure: 16.104567778480828
```



Save your data

After you have completed the data run, save your dataframe with the adsorbed amount versus final pressure to another spreadsheet file. Use the method `to_csv()`. If the dataframe is called `Ar_results`, then

```
Ar_results.to_csv('Argon_isotherm.csv', index=False)
```

will make a simple spreadsheet file. (`index=False` suppresses an index column which is not needed here.)

In [8]:
```
# save your results
Ar_results.to_csv('Argon_isotherm.csv', index=False)
```

Repeat for nitrogen

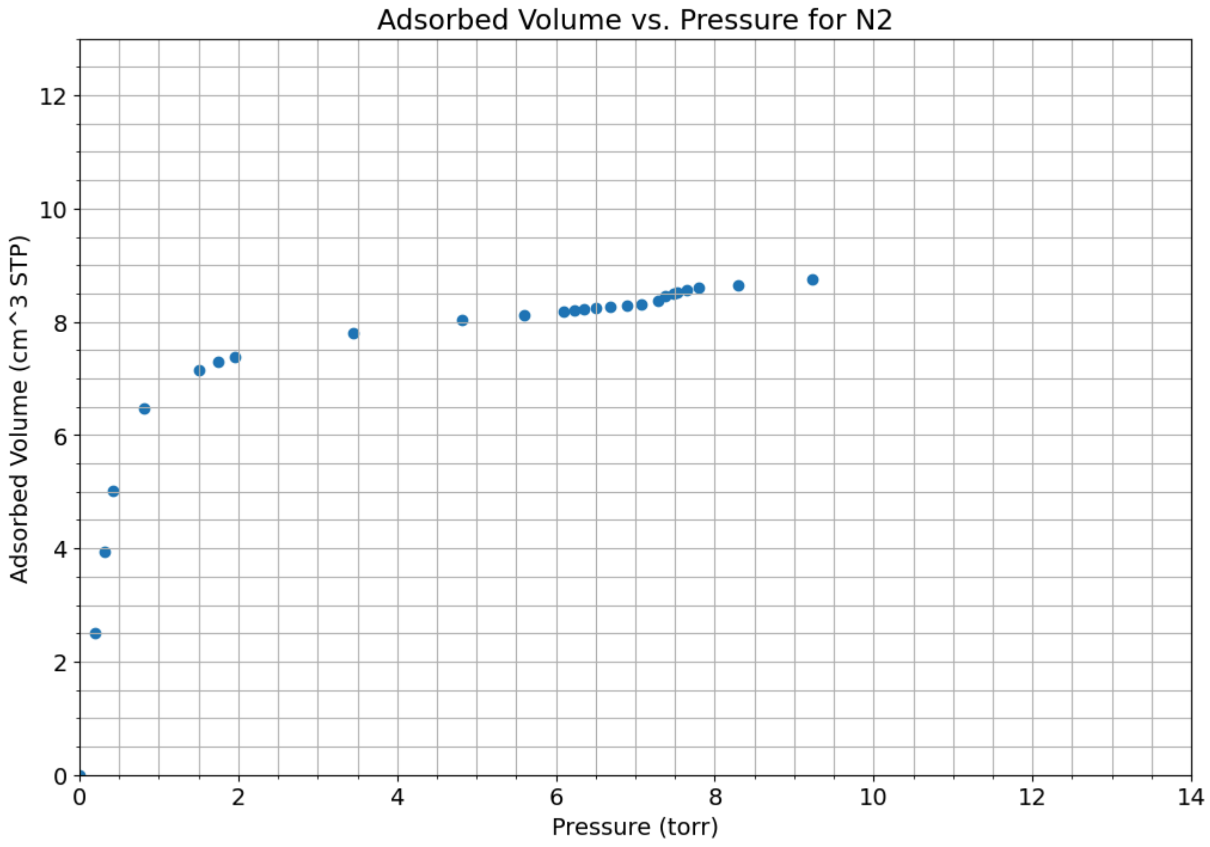
Repeat he above steps for the nitrogen isotherm. You do not need to redo the expected dosing calculation, unless you want.

In [9]:
```
# read in the data and print the datafame
N2InstructionData = pd.read_csv('N2InstructionData.csv')
V_ads=np.array([0.0,  2.5, 3.9, 5.0,  6.5])
P_eq=np.array([0.0,  0.3, 0.5, 0.6, 0.7])

P_c = dose_pressure(V_ads, P_eq)

N2DoseData = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressu
print("N2 instruction data:")
print(N2DoseData)
```

```
N2 instruction data:
   V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0                  0.0                          0.0             24.548573
1                  2.5                          0.3             14.092287
2                  3.9                          0.5             11.256286
3                  5.0                          0.6             15.216429
4                  6.5                          0.7              0.000000
```

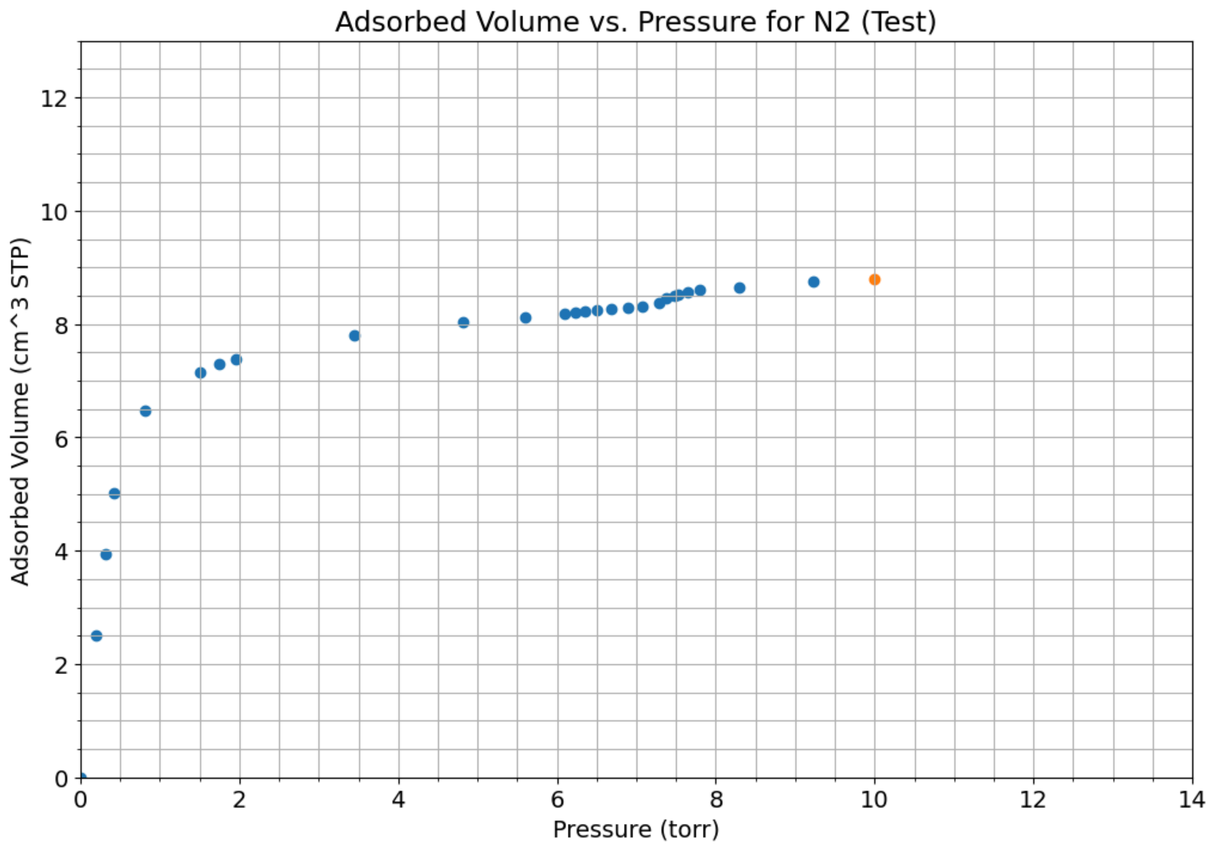
In [10]:
```
# N2 data
P_c = np.array([24.54, 14.10, 10.82, 15.07, 8.41, 3.12, 3.02, 8.05, 7.57, 6.
               7.92, 8.30, 9.03, 10.61])
P_eq = np.array([0, 0.2, 0.32, 0.42, 0.82, 1.50, 1.75, 1.95, 3.45, 4.82, 5.6
                7.64, 7.79, 8.3, 9.23])

# calculate adsorbed volume
V_ads = calc_v_ads(P_eq, P_c)

# store results in dataframe
N2_results = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressu
print("N2 Results:")
print(N2_results)

# Make a plot
plt.scatter(P_eq, V_ads)
plt.title('Adsorbed Volume vs. Pressure for N2')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')
plt.grid(which='both')
plt.minorticks_on()
plt.xlim(0, 14)
plt.ylim(0, 13)
plt.show()
```

```
N2 Results:
    V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0              0.000000                         0.00                 24.54
1              2.513711                         0.20                 14.10
2              3.936553                         0.32                 10.82
3              5.009996                         0.42                 15.07
4              6.469675                         0.82                  8.41
5              7.156896                         1.50                  3.12
6              7.288266                         1.75                  3.02
7              7.390668                         1.95                  8.05
8              7.803772                         3.45                  7.57
9              8.030682                         4.82                  6.87
10             8.129231                         5.60                  6.87
11             8.187834                         6.10                  6.44
12             8.204086                         6.23                  6.53
13             8.217653                         6.35                  6.76
14             8.238239                         6.50                  6.99
15             8.262735                         6.68                  7.28
16             8.292789                         6.90                  7.36
17             8.314176                         7.08                  8.00
18             8.378850                         7.29                  8.08
19             8.449032                         7.37                  8.10
20             8.508617                         7.48                  7.76
21             8.530332                         7.53                  7.92
22             8.554685                         7.64                  8.30
23             8.601177                         7.79                  9.03
24             8.655211                         8.30                 10.61
25             8.758802                         9.23                  0.00
```



In [11]:
```
### This cell is for using existing data to calculate the next dose pressure
### based on the data we took, once the instruction data stops working.

# Existing data:
P_c = N2_results['Dose Pressure (torr)'][0:-1].to_numpy()
P_eq = N2_results['Equilibrium Pressure (torr)'].to_numpy()

# Calculate adsorbed STP volume
V_ads = calc_v_ads(P_eq, P_c)

# Estimate the next point and append to your data arrays
P_eq_next=10
V_ads_next=8.8

P_eq=np.append(P_eq, P_eq_next)
V_ads=np.append(V_ads, V_ads_next)

# Feed the arrays to the dose pressure function, and use the dose pressure t
# then feed data back into the previous cell
P_c = dose_pressure(V_ads, P_eq)
print("Estimated Dose pressure:", P_c[-1])

# Plot for visualization
plt.scatter(P_eq[0:-1], V_ads[:-1])
plt.scatter(P_eq[-1], V_ads[-1])
plt.title('Adsorbed Volume vs. Pressure for N2 (Test)')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')
plt.grid(which='both')
plt.minorticks_on()
plt.xlim(0, 14)
plt.ylim(0, 13)
plt.show()
```

```
Estimated Dose pressure: 10.712452490713385
```



```
In [12]: # Save the final results
         N2_results.to_csv('N2_isotherm.csv', index=False)
```

Use 'Point B' method to find monolayer coverage

The 'point B' method is a standardized way to estimate the monolayer coverage. One fits the data points along the top of the step that are mostly linear in pressure. The point at which this line intersects the knee of the curve is considered point "B". (Point "A" would be where the line intersects the y axis).

Use LMfit. An example is sketched below.

Then fit the line to a slice of the data that is the linear portion. Plot the line on the isotherm, and estimate the line's intersection with the knee by eye.

Use the line parameters to calculate the coverage at the knee.

In [13]:
```
# Set  up the Model

# Import the Linear model.
# You only do this once in a notebook
from lmfit.models import LinearModel

# create an instance of the model
# You only need to do this once
line = LinearModel()
```

In [14]:
```
# Argon first
#

# Select a "slice" of the data set from the results
#
# Here is an example
x_data = Ar_results['Equilibrium Pressure (torr)'][6:]
y_data = Ar_results['V Adsorbed (cc-STP)'][6:]

# Get starting parameters
Ar_params = line.guess(y_data, x=x_data)

# Feed these into the fitter and run it.
Ar_fit = line.fit(y_data, Ar_params, x=x_data)

# Print the results
Ar_fit
```

Out[14]:

Fit Result

Model: Model(linear)

Fit Statistics

fitting method	leastsq
# function evals	4
# data points	6
# variables	2
chi-square	0.01689085
reduced chi-square	0.00422271
Akaike info crit.	-31.2364568
Bayesian info crit.	-31.6529379
R-squared	0.98699217

Parameters

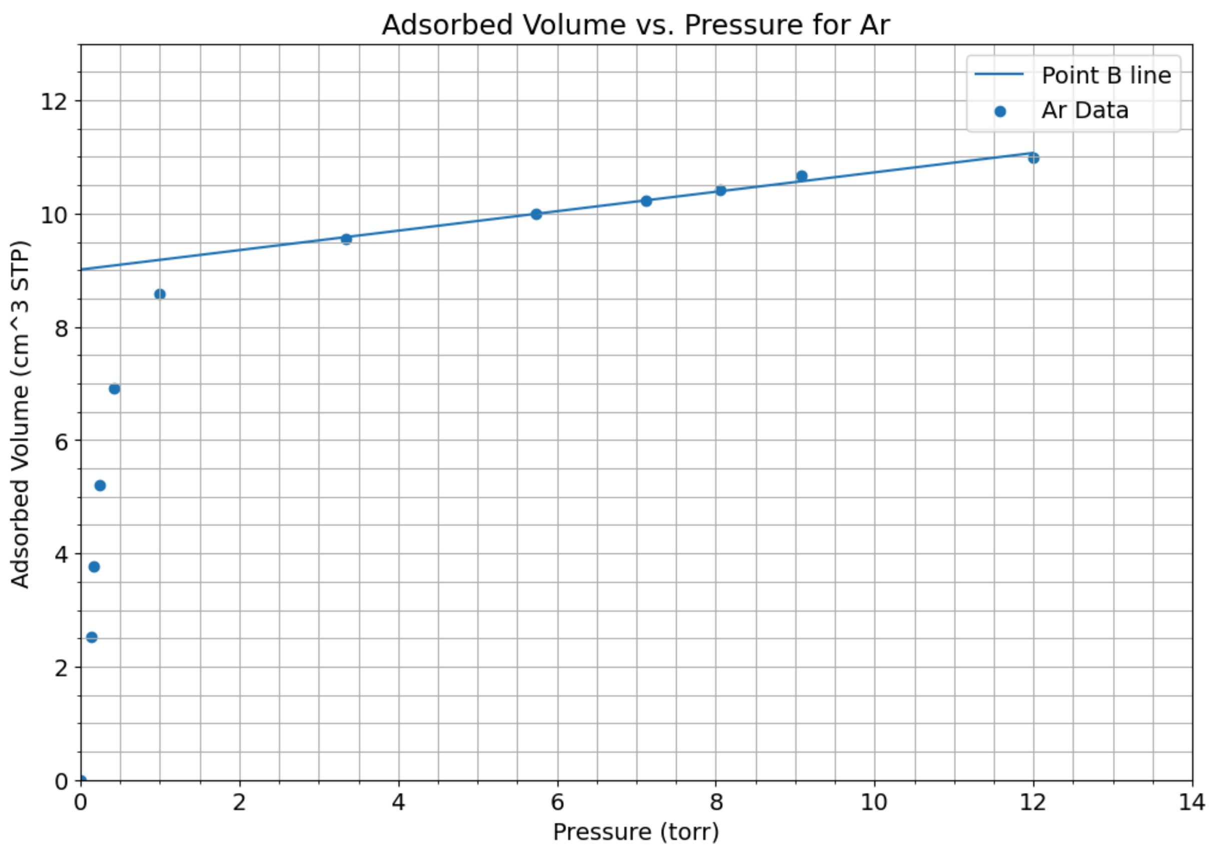
name	value	standard error	relative error	initial value	min	max	vary
slope	0.17166736	0.00985379	(5.74%)	0.17166736026796645	-inf	inf	True
intercept	9.01137293	0.07901555	(0.88%)	9.011372932026031	-inf	inf	True

Correlations (unreported values are < 0.100)

Parameter1	Parameter 2	Correlation
slope	intercept	-0.9420

In [15]:

```
# Make a plot with the line included
# To include the line evaluated across the whole data set, use the eval() me
#
plt.plot(Ar_results['Equilibrium Pressure (torr)'], Ar_fit.eval(x=Ar_results

# plot Ar data
plt.scatter(Ar_results['Equilibrium Pressure (torr)'], Ar_results['V Adsorbe

plt.title('Adsorbed Volume vs. Pressure for Ar')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')

plt.grid(which='both')
plt.minorticks_on()

plt.xlim(0, 14)
plt.ylim(0, 13)
plt.legend()

plt.show()
```



Evaluate the coverage at point 'B'

By eye from the plot, locate where the knee of the data, approximately, intersects the line (or begins to deviat away from it significantly).

Use the `eval()` method to obtain the coverage.

Then estimate an uncertainty for point B, and use eval to estimate an uncertainty in the coverage. Assemble the result into an uncertainty object.

In [16]:
```
# estimate point B and uncertainty
lower=1.75
upper=3
sigma_B_Ar=(upper-lower)/2
B_Ar=unc.ufloat((upper+lower)/2, sigma_B_Ar, 'B_Ar')
print('For Argon:')
print('Point B:   ', B_Ar, 'torr')

# calculate adsorbed volume and molecule count
B_V_ads_Ar=Ar_fit.eval(x=B_Ar)
B_N_ads_Ar=P_std*B_V_ads_Ar / (kB*T_std)
print('V Adsorbed at point B:  ', B_V_ads_Ar, 'cc-STP')
print('Coverage at point B:   ', B_N_ads_Ar)
```

```
For Argon:
Point B:    2.4+/-0.6 torr
V Adsorbed at point B:   9.42+/-0.11 cc-STP
Coverage at point B:    (2.532+/-0.029)e+20
```

Repeat for the nitrogen isotherm

You know what to do now. Avoid the substep in your slice selection.

In [17]:
```
# Select a "slice" of the data set from the results
x_data = N2_results['Equilibrium Pressure (torr)'][8:17]
y_data = N2_results['V Adsorbed (cc-STP)'][8:17]

# Get starting parameters
N2_params = line.guess(y_data, x=x_data)

# Feed these into the fitter and run it.
N2_fit = line.fit(y_data, N2_params, x=x_data)

# Print the results
N2_fit
```

Out[17]:

Fit Result

Model: Model(linear)

Fit Statistics

fitting method	leastsq
# function evals	4
# data points	9
# variables	2
chi-square	9.1432e-04
reduced chi-square	1.3062e-04
Akaike info crit.	-78.7509485
Bayesian info crit.	-78.3564993
R-squared	0.99503967

Parameters

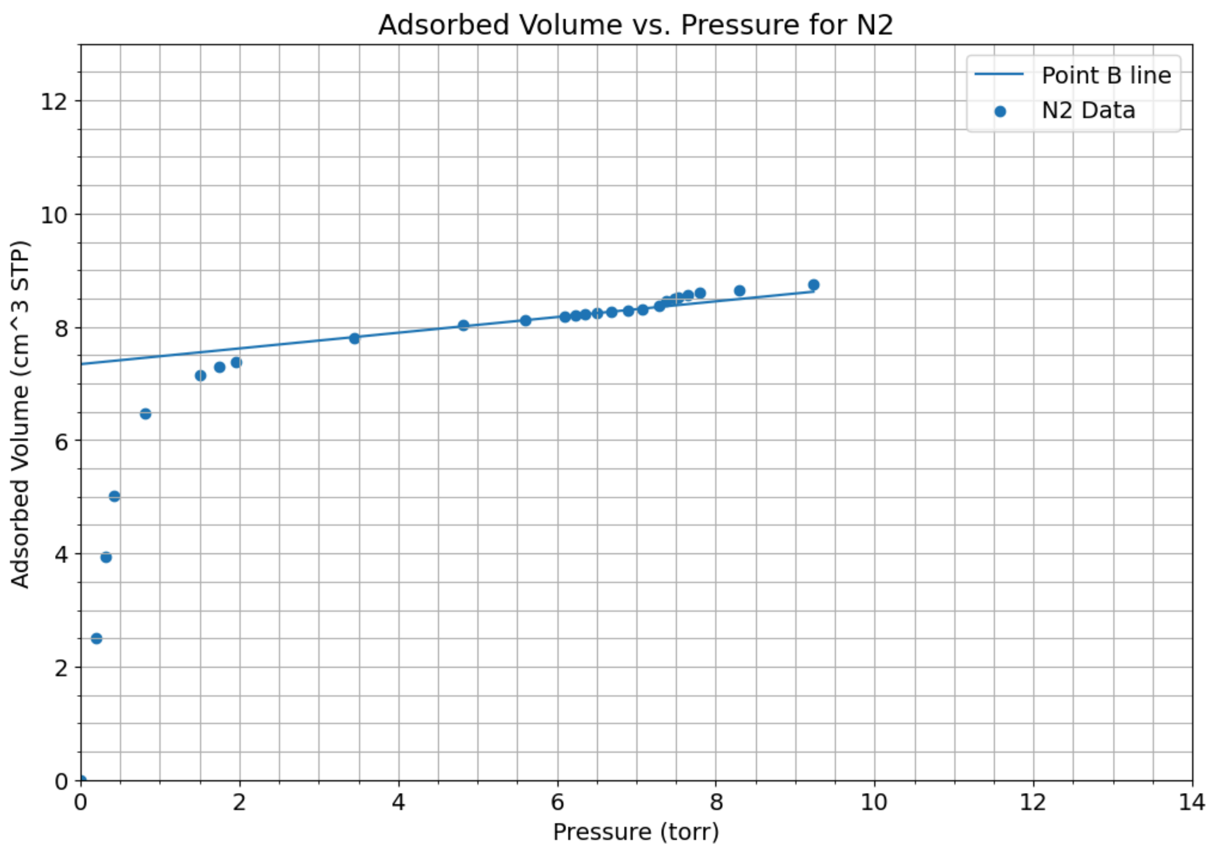
name	value	standard error	relative error	initial value	min	max	vary
slope	0.13855393	0.00369747	(2.67%)	0.13855392548467954	-inf	inf	True
intercept	7.34165865	0.02195504	(0.30%)	7.341658646729307	-inf	inf	True

Correlations (unreported values are < 0.100)

Parameter1	Parameter 2	Correlation
slope	intercept	-0.9848

In [18]:

```
# Make a plot with the line included
# To include the line evaluated across the whole data set, use the eval() me
#
plt.plot(N2_results['Equilibrium Pressure (torr)'], N2_fit.eval(x=N2_results

# plot Ar data
plt.scatter(N2_results['Equilibrium Pressure (torr)'], N2_results['V Adsorbe

plt.title('Adsorbed Volume vs. Pressure for N2')
plt.ylabel('Adsorbed Volume (cm^3 STP)')
plt.xlabel('Pressure (torr)')

plt.grid(which='both')
plt.minorticks_on()

plt.xlim(0, 14)
plt.ylim(0, 13)
plt.legend()

plt.show()
```



In [19]:
```
# Evaluate the coverage at point B and its uncertainty
lower=2.5
upper=3.5
sigma_B_N2=(upper-lower)/2
B_N2=unc.ufloat((upper+lower)/2, sigma_B_N2, 'B_N2')
print('For N2:')
print('Point B:   ', B_N2, 'torr')

# calculate adsorbed volume and molecule count
B_V_ads_N2=N2_fit.eval(x=B_N2)
B_N_ads_N2=B_N2*B_V_ads_N2 / (kB*T_std)
print('V Adsorbed at point B:  ', B_V_ads_N2, 'cc-STP')
print('Coverage at point B:   ', B_N_ads_N2)

# phase transition coverage
PT_V_ads_N2=unc.ufloat(8.5, 0.2, 'PT_N2')
PT_N2=unc.ufloat(7.5, 0.3, 'PT_N2')
PT_N_ads_N2=P_std*PT_V_ads_N2 / (kB*T_std)
print('Phase transition at:    ', PT_N2, 'torr')
print('Phase transition V adsorbed:    ' , PT_V_ads_N2, 'cc-STP')
print('Phase transition coverage:    ' , PT_N_ads_N2)

# calculate coverage
hex_area=5.24*1E-20
surf=1.04*3*hex_area*PT_N_ads_N2
print('Graphite surface area:   ', surf, 'm^2')
```

```
For N2:
Point B:    3.0+/-0.5 torr
V Adsorbed at point B:   7.76+/-0.07 cc-STP
Coverage at point B:    (8.2+/-1.4)e+17
Phase transition at:     7.50+/-0.30 torr
Phase transition V adsorbed:     8.50+/-0.20 cc-STP
Phase transition coverage:     (2.28+/-0.05)e+20
Graphite surface area:    37.4+/-0.9 m^2
```

Recommended: Complete the other calculations in this notebook

These are

- Extract the coverage at the top of the nitrogen isotherm substep and calculate the surface area of the graphite.
- Use the area to calculate the unit cell size of the argon monolayer.
- Look up the bulk unit cell of solid argon (fcc structure) and determine the atomic spacing.
- Evaluate the energy minimum of the Lennard-Jones potential, and estimate the equlibrium spacing from the constants given.
- Make a small table to compare the three different spacing of Ar atoms in different cases: in 3D bulk, in the adsorbed monolayer, and in a simple pair from the L-J potential.

In [20]:
```python
# determine the unit cell area and spacing for Ar at monolayer completion

# calculate coverage ratio
coverage_ratio=PT_N_ads_N2/B_N_ads_Ar
print('Unit cell area ratio (Ar to N2):   ', coverage_ratio)

# calculate Ar unit cell area
cell_area_N2=1.04*3*hex_area
cell_area_Ar=surf*1E20 / B_N_ads_Ar # #cell_area_N2*coverage_ratio*10**20
print('Unit cell area of Ar:   ', cell_area_Ar, 'A^2')

# calculate Ar spacing
cell_spacing_Ar=(2*cell_area_Ar / (3**0.5))**0.5
print('Unit cell spacing for Ar:   ', cell_spacing_Ar, 'A')
```

```
Unit cell area ratio (Ar to N2):    0.902+/-0.024
Unit cell area of Ar:    14.8+/-0.4 A^2
Unit cell spacing for Ar:    4.13+/-0.05 A
```

In [21]:
```python
### This section is for calculations relating to the individual report
###
###
# Create arrays to hold the extracted results
V_ads = np.array([0, PT_V_ads_N2.n])
P_eq = np.array([0, PT_N2.n])
```

```
# Find dose pressure
P_c = dose_pressure(V_ads, P_eq)

Dose2PhaseTrans = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium P
print('Dose to reach N2 phase transition:')
print(Dose2PhaseTrans)
print('\n')



# Create arrays to hold the extracted results
V_ads = np.array([0, B_V_ads_Ar.n])
P_eq = np.array([0, B_Ar.n])

# Find dose pressure
P_c = dose_pressure(V_ads, P_eq)

Dose2B = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressure (
print('Dose to reach Ar knee:')
print(Dose2B)
```

```
Dose to reach N2 phase transition:
   V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0                  0.0                          0.0             92.595019
1                  8.5                          7.5              0.000000


Dose to reach Ar knee:
   V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0             0.000000                        0.000             94.243728
1             9.419083                        2.375              0.000000
```

In [106…

```
# change sample volume to incorrect value
v_s=16.8

# N2 manufactured data
P_c = Dose2PhaseTrans['Dose Pressure (torr)'][0:1].to_numpy()
P_eq = Dose2PhaseTrans['Equilibrium Pressure (torr)'].to_numpy()

# calculate adsorbed volume
V_ads = calc_v_ads(P_eq, P_c)

# store results in dataframe
N2_results = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressu
print("N2 results for wrong sample volume:")
print(N2_results)
print('\n')


# Ar manufactured data
P_c = Dose2B['Dose Pressure (torr)'][0:1].to_numpy()
P_eq = Dose2B['Equilibrium Pressure (torr)'].to_numpy()

# calculate adsorbed volume
V_ads = calc_v_ads(P_eq, P_c)
```

```
# store results in dataframe
Ar_results = pd.DataFrame({'V Adsorbed (cc-STP)': V_ads, 'Equilibrium Pressu
print("Ar results for wrong sample volume:")
print(Ar_results)
```

```
N2 results for wrong sample volume:
   V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0             0.000000                          0.0             92.595019
1             8.664384                          7.5              0.000000


Ar results for wrong sample volume:
   V Adsorbed (cc-STP)  Equilibrium Pressure (torr)  Dose Pressure (torr)
0             0.000000                        0.000             94.243728
1             9.471138                        2.375              0.000000
```

In [109…

```
print('Ar at knee to N2 phase trans for correct v_s:')
print(B_V_ads_Ar.n / PT_V_ads_N2.n)
print('\n')

print('Ar at knee to N2 phase trans for incorrect v_s:')
Ar_VB=Ar_results['V Adsorbed (cc-STP)'][1]
N2_VPT=N2_results['V Adsorbed (cc-STP)'][1]
print(Ar_VB / N2_VPT)
```

```
Ar at knee to N2 phase trans for correct v_s:
1.1081274014897002


Ar at knee to N2 phase trans for incorrect v_s:
1.0931114635204264
```

In [ ]: