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 [18]:
```
# Usually import packages via a handle to the functions in them using import ... as
#
import numpy as np
import matplotlib.pyplot as plt
import matplotlib as mpl
import pandas as pd
import uncertainties as unc
%matplotlib inline
```

In [19]:
```
# 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 [20]:
```
# Create your function here
def expected_dose(Pf_want, Pf_last, Vstp_add, T = 295.5, vs = 34.8,
                  vc = 85.1, T0 = 273, P0 = 760):
    return (P0*T/T0/vc) * Vstp_add + Pf_want + (vs/vc) * (Pf_want - Pf_last)

print(expected_dose(0.3, 0, 2.5, 295), "Torr")
print(expected_dose(0.5, 0.3, 1.2, 295), "Torr")

def dose_chopper(eq_Ps,Vstps, T = 295.5):
    doses = []
    for i in range(1, len(eq_Ps)):
        dose = expected_dose(eq_Ps[i], eq_Ps[i - 1], Vstps[i] - Vstps[i - 1], T)
        print(f"Expected calibration chamber pressure for dose {i}: {dose:.2f} Torr
        doses.append(dose)
    return doses
```

```
24.548572892051155 Torr
12.162215105693368 Torr
```

Read the graph showing example data (Figure 5) and extract the adsorbed amount V_{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 [22]:
```
# Create arrays to hold the extracted results
eq_Ps = np.array([0, 0.3, 0.5, 0.5, 0.6])
Vstps = np.array([0, 2.5, 3.6, 5.1, 6.9])

# Feed the arrays to your function above, and print a table
doses_array = dose_chopper(eq_Ps, Vstps)
```

```
Expected calibration chamber pressure for dose 1: 24.59 Torr
Expected calibration chamber pressure for dose 2: 11.22 Torr
Expected calibration chamber pressure for dose 3: 15.00 Torr
Expected calibration chamber pressure for dose 4: 18.04 Torr
```

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 [23]:
```
# Create your function here
def V_added(Pi, Pf, Pf_last, T = 294.65, vs = 34.8, vc = 85.1, T0 = 273, P0 = 760):
    return (T0/T/P0) * (vc * (Pi - Pf) - vs * (Pf - Pf_last))

#Takes our measured lab values and uses the previous points to calculate the Vstp.
def plotting(measured_Pi, measured_Pf, csv_filepath, xlim = 15, ylim = 15):
    lab_data = pd.read_csv(csv_filepath)
    curr_Pf_array = lab_data['Pf'].to_numpy()
```

```
    curr_Vstp_array = lab_data['Vstp'].to_numpy()

    #Adding the values to the array
    curr_Pf_array = np.append(curr_Pf_array, measured_Pf)
    Vadd = Vadded(measured_Pi, curr_Pf_array[-1], curr_Pf_array[-2])
    Vstp = curr_Vstp_array[-1] + Vadd
    curr_Vstp_array = np.append(curr_Vstp_array, Vstp)

    temp_df = pd.DataFrame({'Pf': curr_Pf_array,
                        'Vstp': curr_Vstp_array})
    temp_df.to.csv(csv_filepath, index = False)

    print(f"Pf_last: {measured_Pf:.f} Torr, Vstp_last: {Vstp:.2f} cc")

    plt.plot(curr_Pf_array, curr_Vstp_array, 'o')
    plt.title('Pf vs Vstp Isotherm')
    plt.xlabel('Pf [Torr]')
    plt.ylabel('Adsorbed Vstp [cc]')
    plt.xlim(0, xlim)
    plt.ylim(0, ylim)
    plt.show
```

Recommended: open a spreadsheet to save the data to a separate file. Then read in the spreadsheet with Pandas `pd.read_csv()`:

```
Ar_data = pd.read_csv('Ar_adsorption_data.csv')
```

In [ ]:
```
# Read in the data and print the dataframe
```

Build another dataframe from your function and the final pressure at each dose.

In [ ]:
```
# Make a results dataframe from the calculation and pressure values
```

Make a plot

Plot the adsorbed amount V_{ads} versus pressure P

In [24]:
```
# Last measurement from plotting the n2 isotherm

estimated_dose = expected_dose(Pf_want = 9.5, Pf_last = 8.27, Vstp_add = 0.01)
print(f"est dose: {estimated_dose}")
print(f"voltmeter reading: {(estimated_dose - 0.27)/100:.4f}")
```

```
est dose: 10.099651863999691
voltmeter reading: 0.0983
```

In [ ]:
```
#Plot

#plotting(9, 8.27, 'N2_adsorption_data.csv)
#This would be the the code used for plotting
#but the experiment has already been conducted
```

Repeat the above cells to build your plot as you go:

1. Calculate the dose
2. Carry out the dosing for that step:
   - Record the initial pressure in the spreadsheet
   - Measure and record the final pressure in the spreadsheet
3. Read in the sheet and calculate the adsorbed amount
4. Plot the whole data set

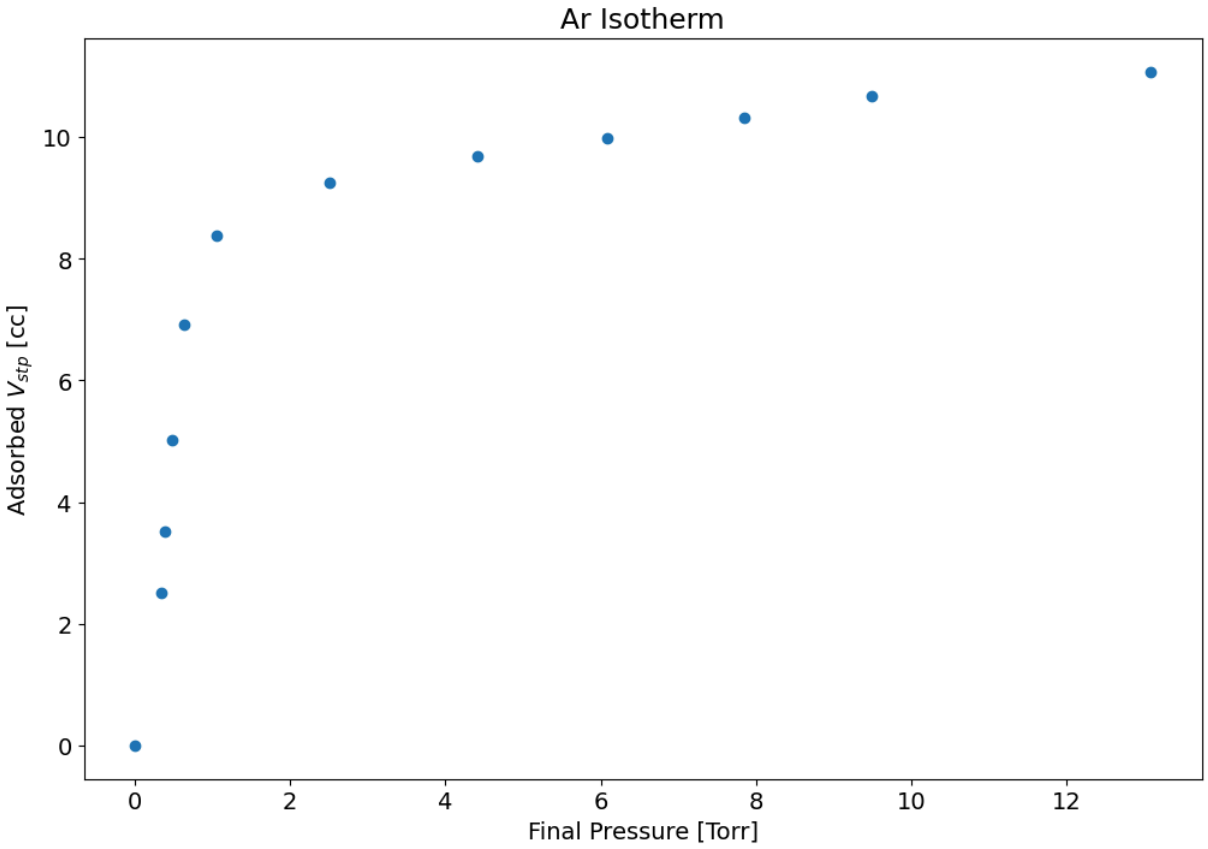
Save your data

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 [25]:
```
# save your results

def simple_plot(csv_filepath):
    lab_data = pd.read_csv(csv_filepath)
    curr_Pf_array = lab_data['Pf'].to_numpy()
    curr_Vstp_array = lab_data['Vstp'].to_numpy()
    plt.plot(curr_Pf_array, curr_Vstp_array, 'o')
    plt.xlabel('Final Pressure [Torr]')
    plt.ylabel(r'Adsorbed $V_{stp}$ [cc]')
    if 'Ar' in csv_filepath:
        plt.title('Ar Isotherm')
    else:
        plt.title(r'$N_{2}$ Isotherm')

    plt.show()

simple_plot('Ar_isotherm.csv')

Ar_data = pd.read_csv('Ar_isotherm.csv')
Ar_data
```



Out[25]:

	Pf	Vstp
0	0.00	0.000000
1	0.34	2.499346
2	0.39	3.511862
3	0.48	5.030001
4	0.64	6.923843
5	1.06	8.374033
6	2.51	9.255570
7	4.41	9.695768
8	6.08	9.991142
9	7.84	10.317972
10	9.48	10.688279
11	13.08	11.080217

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 [26]:
```
# read in the data and print the dataframe
n2_data = pd.read_csv("N2_isotherm.csv")
n2_data
```

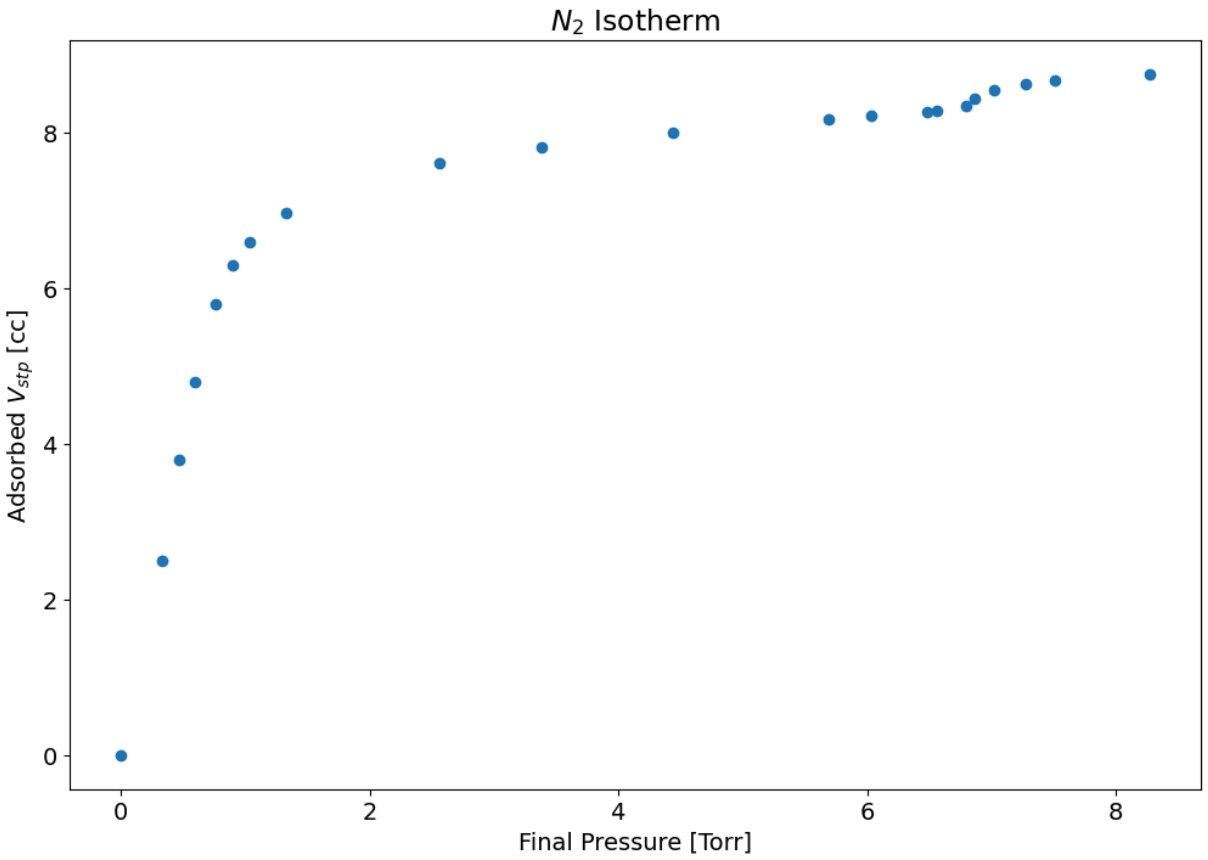
Out[26]:

	Pf	Vstp
0	0.00	0.000000
1	0.33	2.502882
2	0.47	3.803107
3	0.59	4.806429
4	0.76	5.799331
5	0.90	6.295523
6	1.03	6.590871
7	1.33	6.973417
8	2.56	7.608034
9	3.38	7.813937
10	4.44	8.007582
11	5.69	8.175530
12	6.03	8.221279
13	6.48	8.275847
14	6.56	8.289053
15	6.79	8.342580
16	6.86	8.449581
17	7.02	8.549652
18	7.27	8.630342
19	7.51	8.679296
20	8.27	8.761173

In [ ]:
```
# Make a results dataframe from the calculation and pressure values
```

In [27]:
```
# Make a plot
```

```
simple_plot("N2_isotherm.csv")
```



In [ ]:
```
# Save the final results
```

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 [28]:
```
# 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 [39]:
```
# Argon first
#

# Select a "slice" of the data set from the results
#
# Here is an example
indices = list(range(6,12))
x_data = Ar_data['Pf'][indices]
y_data = Ar_data['Vstp'][indices]

# 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[39]:

Fit Result

Model: Model(linear)

Fit Statistics

fitting method	leastsq
# function evals	4
# data points	6
# variables	2
chi-square	0.03877772
reduced chi-square	0.00969443
Akaike info crit.	-26.2500132
Bayesian info crit.	-26.6664942
R-squared	0.98246985

Parameters

name	value	standard error	relative error	initial value	min	max	vary
slope	0.17469738	0.01166782	(6.68%)	0.17469738465969797	-inf	inf	True
intercept	8.90784692	0.09348063	(1.05%)	8.90784691656239	-inf	inf	True

Correlations (unreported values are < 0.100)

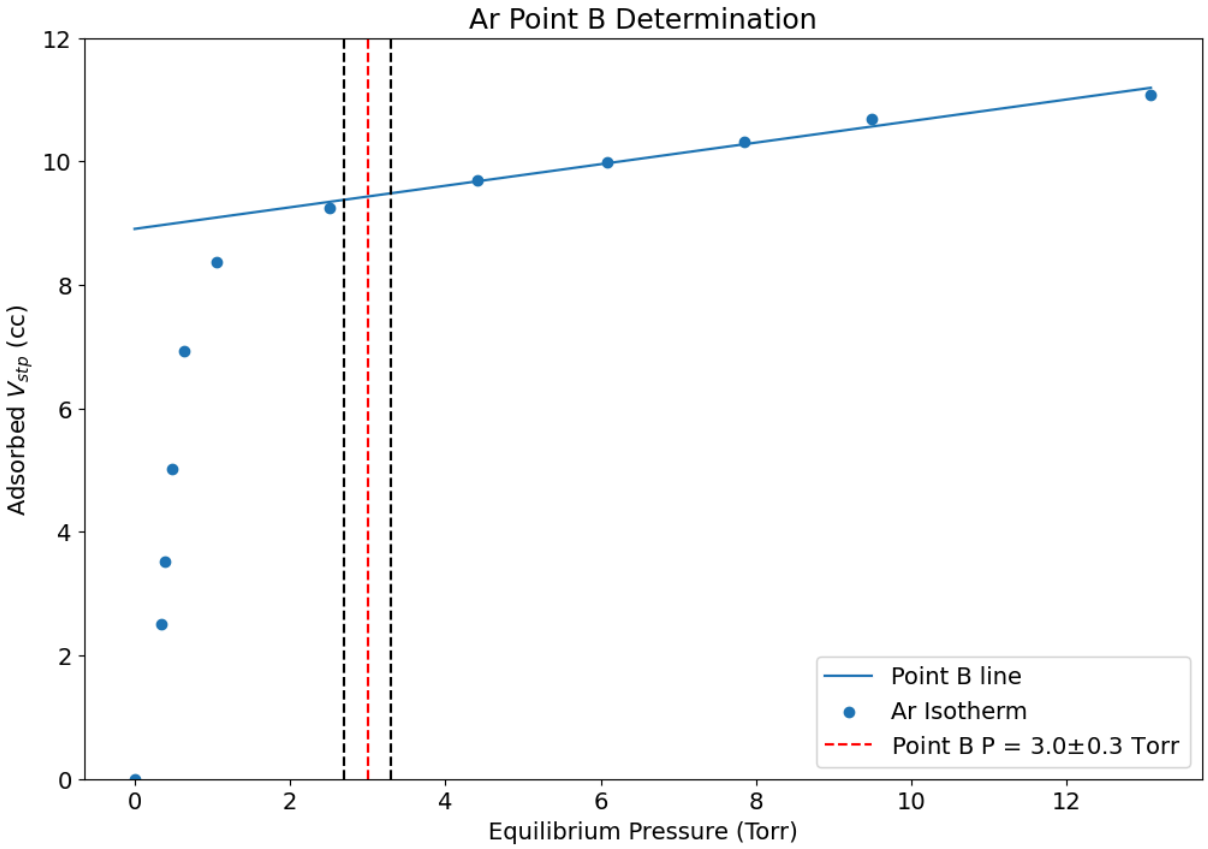
Parameter1	Parameter 2	Correlation
slope	intercept	-0.9028

In [40]:

```
# Make a plot with the line included
# To include the line evaluated across the whole data set, use the eval() method.
#
plt.plot(Ar_data['Pf'],Ar_fit.eval(x=Ar_data['Pf']),'-',label='Point B line')
# Add in he rest of the plot commands here, including the data.
plt.scatter(Ar_data['Pf'], Ar_data['Vstp'], label = 'Ar Isotherm')

pB = 3.0
unky = 0.3

plt.axvline(pB, color = 'r', linestyle = '--',
            label = f'Point B P = {pB}' + r'$\pm$' + str(unky) + ' Torr')
plt.axvline(pB + unky, color = 'black', linestyle = '--')
plt.axvline(pB - unky, color = 'black', linestyle = '--')

plt.title('Ar Point B Determination')
plt.xlabel('Equilibrium Pressure (Torr)')
```

```
plt.ylabel(r'Adsorbed $V_{stp}$ (cc)')
plt.ylim(0,12)
plt.legend()
```

Out[40]: <matplotlib.legend.Legend at 0x28963f32d10>



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 [46]:
```
# You code this
coverage_nom = Ar_fit.eval(x = pB)
coverage_unc = Ar_fit.eval(x = pB + unky) - coverage_nom

coverage_B = unc.ufloat(coverage_nom, coverage_unc)
print(f"Coverage at Point B: {coverage_B:.3f}")
```

```
Coverage at Point B: 9.432+/-0.052
```

Repeat for the nitrogen isotherm

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

In [48]:
```
# Select a "slice" of the data set from the results
indices = list(range(9, 15))
x_data2 = n2_data['Pf'][indices]
y_data2 = n2_data['Vstp'][indices]

# Get starting parameters
n2_params = line.guess(y_data2, x = x_data2)

# Feed these into the fitter and run it.
n2_fit = line.fit(y_data2, n2_params, x = x_data2)

# Print the results
n2_fit
```

Out[48]:

Fit Result

Model: Model(linear)

Fit Statistics

fitting method	leastsq
# function evals	4
# data points	6
# variables	2
chi-square	9.2963e-04
reduced chi-square	2.3241e-04
Akaike info crit.	-48.6348814
Bayesian info crit.	-49.0513625
R-squared	0.99459058

Parameters

name	value	standard error	relative error	initial value	min	max	vary
slope	0.14626488	0.00539341	(3.69%)	0.14626488172421812	-inf	inf	True
intercept	7.33631964	0.02994021	(0.41%)	7.336319640918806	-inf	inf	True

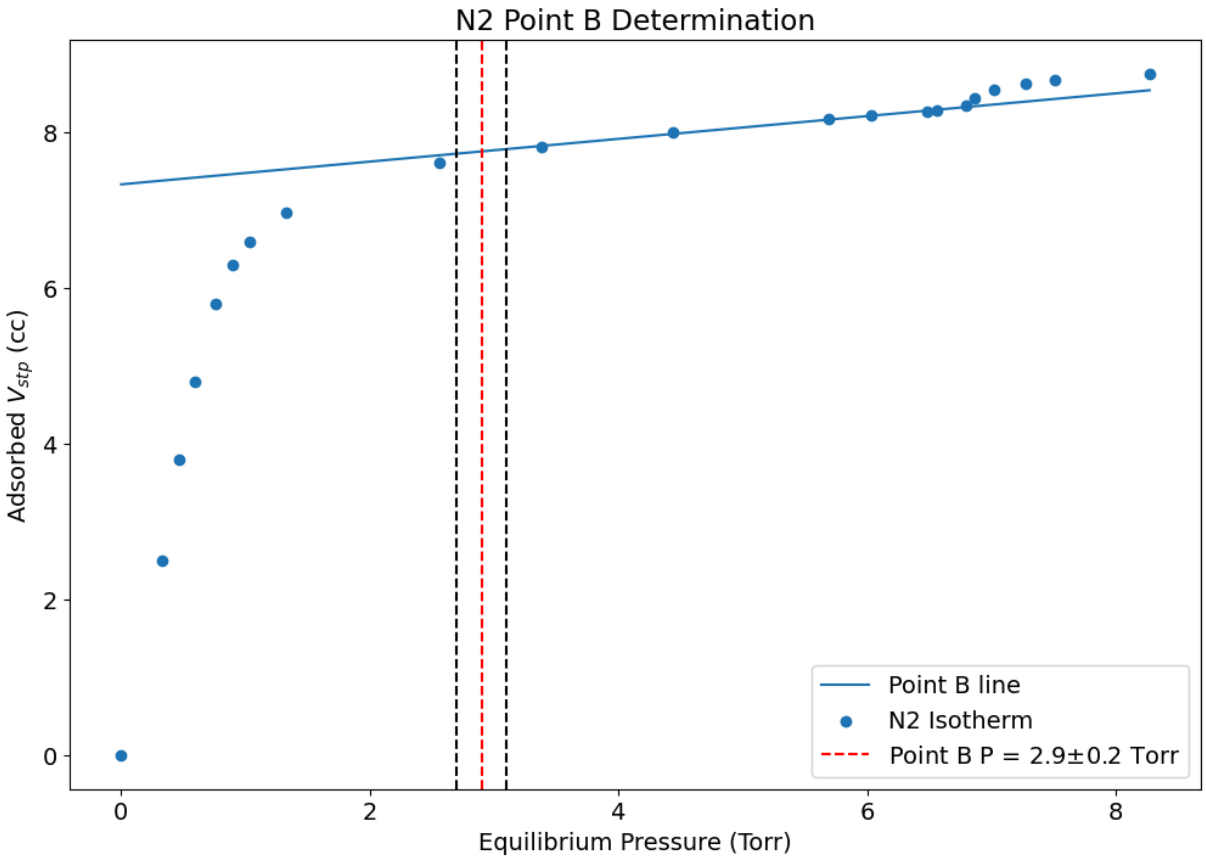
Correlations (unreported values are < 0.100)

Parameter1	Parameter 2	Correlation
slope	intercept	-0.9782

In [51]:
```
# Make a plot with the line included
# To include the line evaluated across the whole data set, use the eval() method.
```

```
#
plt.plot(n2_data['Pf'],n2_fit.eval(x=n2_data['Pf']),'-',label='Point B line')
# Add in he rest of the plot commands here, including the data.
plt.scatter(n2_data['Pf'], n2_data['Vstp'], label = 'N2 Isotherm')

pB2 = 2.9
unky2 = 0.2

plt.axvline(pB2, color = 'r', linestyle = '--',
            label = f'Point B P = {pB2}' + r'$\pm$' + str(unky2) + ' Torr')
plt.axvline(pB2 + unky2, color = 'black', linestyle = '--')
plt.axvline(pB2 - unky2, color = 'black', linestyle = '--')

plt.title('N2 Point B Determination')
plt.xlabel('Equilibrium Pressure (Torr)')
plt.ylabel(r'Adsorbed $V_{stp}$ (cc)')
plt.legend()
```

Out[51]: <matplotlib.legend.Legend at 0x2896a7ecd90>



In [53]:
```
# Evaluate the coverage at point B and its uncertainty
n2_coverage_nom = Ar_fit.eval(x = pB2)
n2_coverage_unc = Ar_fit.eval(x = pB2 + unky2) - n2_coverage_nom

n2_coverage_B = unc.ufloat(n2_coverage_nom, n2_coverage_unc)
print(f"N2 Coverage at Point B: {n2_coverage_B:.2u}")
```

```
N2 Coverage at Point B: 9.414+/-0.035
```

Recommended: Complete the other calculations in this notebook

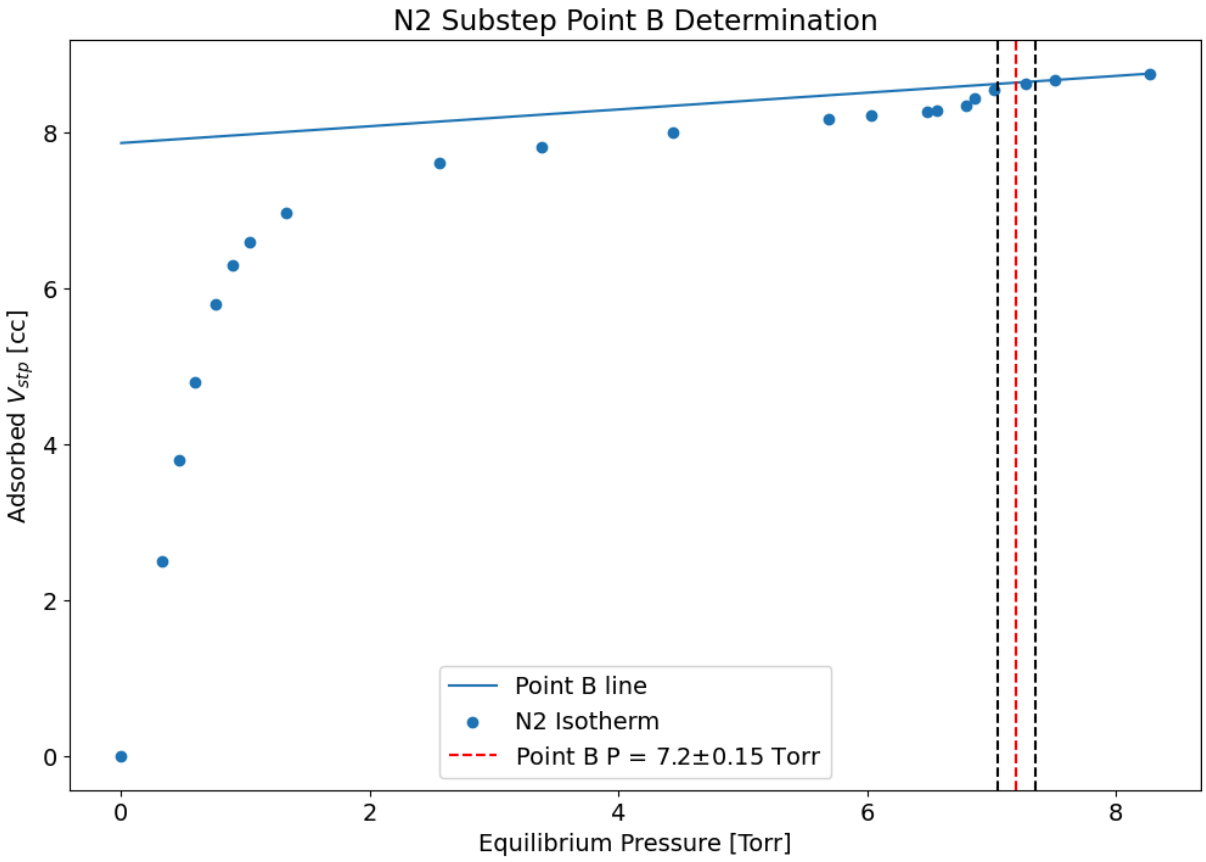
These are

- 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 [ ]:
```
# Add code cells as needed.
```

In [54]:
```
#N2 Substep Point B

indices = list(range(19,21))
x_data3 = n2_data['Pf'][indices]
y_data3 = n2_data['Vstp'][indices]

n2_params2 = line.guess(y_data3, x_data3)

n2_fit2 = line.fit(y_data3, n2_params2, x=x_data3)

# Print the results
n2_fit2

plt.plot(n2_data['Pf'],n2_fit2.eval(x=n2_data['Pf']),'-',label='Point B line')
plt.scatter(n2_data['Pf'],n2_data['Vstp'],label='N2 Isotherm')

pB3 = 7.2
unky3 = 0.15
plt.axvline(pB3, color='r', linestyle='--',
            label=f'Point B P = {pB3}' + r'$\pm$' + str(unky3) + ' Torr')
plt.axvline(pB3 + unky3, color='black', linestyle='--')
plt.axvline(pB3 - unky3, color='black', linestyle='--')

plt.title('N2 Substep Point B Determination')
plt.xlabel('Equilibrium Pressure [Torr]')
plt.ylabel(r'Adsorbed $V_{stp}$ [cc]')
plt.legend()
```

Out[54]:
```
<matplotlib.legend.Legend at 0x2896aee8410>
```



In [57]:
```
#Covererage at Point B

n2_coverage_nom2 = n2_fit2.eval(x = pB3)
n2_coverage_unc2 = n2_fit2.eval(x = pB3+unky3) - n2_coverage_nom2

n2_pressure2_B = unc.ufloat(pB3, unky3)
n2_coverage2_B = unc.ufloat(n2_coverage_nom2, n2_coverage_unc2)

#Print results
print(f"Pressure at completion: {pB3:.2f}+/-{unky3:.2f} Torr")
print(f"n2 Coverage at Point B: {n2_coverage2_B:.2u} cc")
print(f"n2 pressure at Point B: {n2_pressure2_B:.2u} cc")
```

```
Pressure at completion: 7.20+/-0.15 Torr
n2 Coverage at Point B: 8.646+/-0.016 cc
n2 pressure at Point B: 7.20+/-0.15 cc
```

In [60]:
```
#Surface area of the graphite in meters

hex_a = 5.24E-20
n_ads = 101325 * n2_coverage2_B * (1E-6 / (273 * 1.381E-23))
SA_total = 1.04 * 3 * hex_a * n_ads

print(f"Total Surface Area: {SA_total:.2u} m^2")
```

```
Total Surface Area: 37.989+/-0.071 m^2
```

In [61]:
```
#Argon monolayer

#molecules/m^2
```

```
n2_density = 1/15.7E-20
#molecules/angstrom^2
n2_ang_dense = 1/15.7

Ar_density = (coverage_B/n2_coverage2_B) * n2_density
Ar_density_ang = (coverage_B/n2_coverage2_B) * n2_ang_dense

print(f"Ar density: {Ar_density:.2u} molecules/m^2")
print(f"Ar density: {Ar_density*1E-20:.2u} molecules/angstrom^2")
print(f"Ar density: {Ar_density_ang:.2u} molecules/angstrom^2")
```

```
Ar density: (6.949+/-0.041)e+18 molecules/m^2
Ar density: 0.06949+/-0.00041 molecules/angstrom^2
Ar density: 0.06949+/-0.00041 molecules/angstrom^2
```

In [62]:
```
#Fcc Argon
#Lattice constant in angstroms
lat_const = 5.26
r = (np.sqrt(2) * lat_const) / 2
print(f"Ar-Ar spacing: {r:.2f} angstroms")
```

```
Ar-Ar spacing: 3.72 angstroms
```