
adsorption_isotherm_fitting

Release v0.0.3

Robert F. DeJaco

Oct 25, 2020

CONTENTS:

1	Installation	1
1.1	Docker	1
1.2	Singularity	1
1.3	Scipy Only	1
2	CO2/N2 Unary Example	3
2.1	Initialization	3
2.2	CO2	3
2.3	N2	5
2.4	Comparison to scipy	6
3	H2S/CH4 Example	7
3.1	Initialization	7
3.2	Solution	8
3.3	Comparison to scipy	11
4	Unary Isotherms	13
4.1	Langmuir	13
4.2	Modules	14
5	Binary Isotherms	17
5.1	Binary Langmuir	17
5.2	Modules	18
6	Adsorption Equilibria Rules	21
7	Real Adsorbed Solution Theory	23
8	Data Sets	25
8.1	CO2 and N2 on BEA	25
8.2	H2S and CH4 on MFI	29
8.2.1	Fugacity coefficients and fugacities	34
9	References	39
10	Indices and tables	41
	Bibliography	43
	Index	45

INSTALLATION

1.1 Docker

```
docker pull dejac001/isotherm-fitting-users:0.0.4
docker run -ti -v $PWD:/home/pyomo/shared/ dejac001/isotherm-fitting-users:0.0.4 #_
↳ run interactively inside container (ubuntu-based)
```

1.2 Singularity

```
module load singularity
singularity pull docker://dejac001/isotherm-fitting-users:0.0.4
mv isotherm-fitting-users_0.0.4.sif /path/to/shared/directory/isotherm-fitting-users_
↳ 0.0.4.sif
singularity exec -B $PWD:/home/pyomo/shared /path/to/shared/directory/isotherm-
↳ fitting-users_0.0.4.sif python3 path/to/input/file.py
```

1.3 Scipy Only

```
pip3 install Pyomo chem-util matplotlib pandas numpy realgas>=1.0.2
python3 -m pip3 install https://github.com/dejac001/adsorption_isotherm_fitting/
↳ archive/v0.0.4.tar.gz
```


CO2/N2 UNARY EXAMPLE

In this example, we fit temperature-dependent unary data from [PXSL14].

2.1 Initialization

We first load the necessary packages

```
>>> import pyomo.environ as pyo
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> from isotherm_models.unaryisotherm import LangmuirUnary
```

2.2 CO2

We first get the data from the data file

```
>>> data = pd.read_csv('data_sets/CO2_BEA.csv')
```

Using pandas, we can easily take a peek at the data we have input from our .csv file

```
>>> data.head()
   P [atm]  Q [mmol/g]  T [K]  adsorbate
0  0.029814  0.096491  273.0         CO2
1  0.055856  0.175439  273.0         CO2
2  0.109177  0.328947  273.0         CO2
3  0.246821  0.719298  273.0         CO2
4  0.313781  0.903509  273.0         CO2
```

Before solving the model, we convert the partial pressures to si units

```
>>> P_i = data['P [atm]']*101325 # convert to Pa -- si units
```

so that we can create the model

```
>>> co2_model = LangmuirUnary(P_i, data['Q [mmol/g]'], data['T [K]'], name='CO2')
```

and solve it

```
>>> co2_model.solve()
```

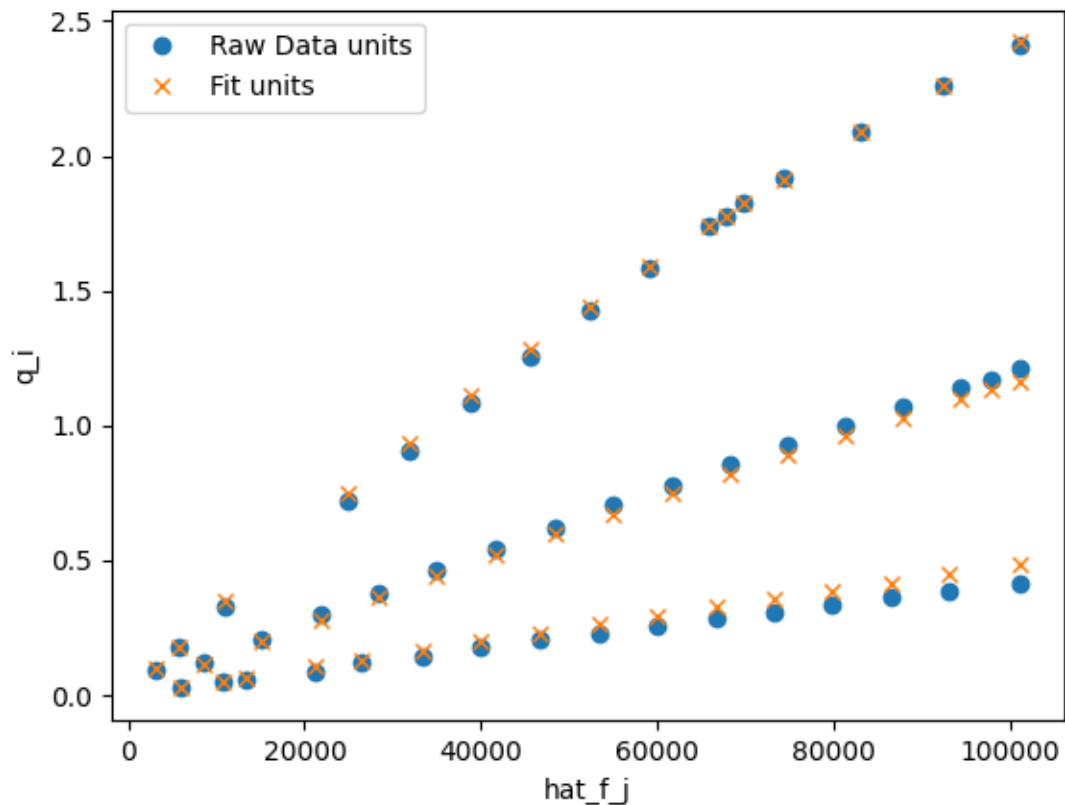
We then take a look at the results

```
>>> co2_model.get_R2_pyomo()
0.99796
>>> co2_model.get_objective()
0.007229102
>>> co2_model.dH_i.display()
dH_i : Size=1
      Key : Value
      None : -20780.90809523844
>>> co2_model.q_mi.display()
q_mi : Size=1
      Key : Value
      None : 8.95582798469325
>>> co2_model.k_i_inf.display()
k_i_inf : Size=1
      Key : Value
      None : 3.8656918601559114e-10
```

And save the results to a file

```
>>> fig = plt.figure()
>>> fig, ax = co2_model.plot_unary(fig=fig)
>>> _ = ax.legend()
>>> fig.savefig('docs/source/CO2_example.png')
```

which looks like



2.3 N2

We repeat a similar approach for the N2 isotherms, first formatting the data for input to the model

```
>>> data = pd.read_csv('data_sets/N2_BEA.csv')
```

Using pandas, we can easily take a peek at the data we have input from our .csv file

```
>>> data.head()
   P [atm]  Q [mmol/g]  T [K]  adsorbate
0  0.525470  0.070175  303.0         N2
1  0.592387  0.078947  303.0         N2
2  0.656824  0.083333  303.0         N2
3  0.722502  0.092105  303.0         N2
4  0.788179  0.100877  303.0         N2
```

Before solving the model, we convert the partial pressures to si units

```
>>> P_i = data['P [atm]']*101325 # convert to Pa -- si units
```

Instantiating (creating) the model

```
>>> n2_model = LangmuirUnary(P_i, data['Q [mmol/g]'], data['T [K]'], name='N2')
```

Solving it

```
>>> n2_model.solve()
```

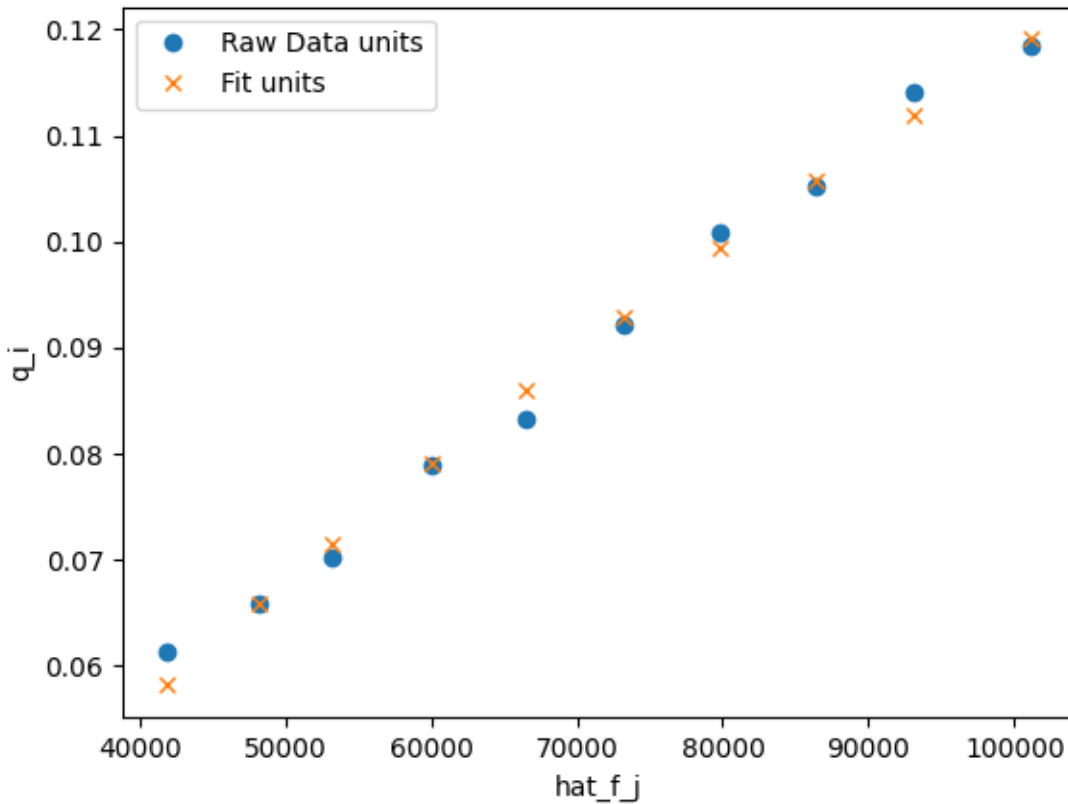
We then take a look at the results

```
>>> n2_model.get_R2_pyomo()
0.99262
>>> n2_model.get_objective()
0.00194249
>>> n2_model.dH_i.display()
dH_i : Size=1
      Key : Value
      None : -12557.526993112784
>>> n2_model.q_mi.display()
q_mi : Size=1
      Key : Value
      None : 0.45280441671269905
>>> n2_model.k_i_inf.display()
k_i_inf : Size=1
          Key : Value
          None : 2.412336128388879e-08
```

And save the results to a file

```
>>> fig = plt.figure()
>>> fig, ax = n2_model.plot_unary(fig=fig)
>>> _ = ax.legend()
>>> fig.savefig('docs/source/N2_example.png')
```

which looks like



2.4 Comparison to scipy

```
>>> import numpy as np
>>> popt, pcov = co2_model.solve_scipy()
>>> popt
array([ 3.71939309, -10.14817759, -7.28715595])
>>> popt - np.array(list(map(pyo.value, [co2_model.q_mi_star, co2_model.A_i, co2_
↪model.H_i_star])))
array([3.28355311e-05, 3.67969745e-05, 3.88858220e-05])
```

H2S/CH4 EXAMPLE

In this example, we fit temperature-dependent binary data from [STS15] and compare the results to using the extended Langmuir combining rule.

3.1 Initialization

First, we import the necessary packages

```
>>> import pyomo.environ as pyo
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> from isotherm_models.unaryisotherm import LangmuirUnary
>>> from isotherm_models.binaryisotherm import BinaryLangmuir
```

First, we grab the data for adsorption of H2S:

```
>>> df = pd.read_csv('data_sets/CH4_H2S_MFI_binary_with_fugacity.csv')
>>> hat_f_i, hat_f_j, q_i, T = df['fugacity H2S [Pa]'], df['fugacity CH4 [Pa]'], df[
↪ 'Q H2S [mmol/g]'], df['T [K]']
```

Here, we are going to fit the loading of H2S, q_i , as a function of the (mixture) fugacities of H2S, $\hat{f}_i$, and CH4, $\hat{f}_j$. Since the data file includes both binary and unary data (including CH4 unary data where H2S is not present), we need to find the data points where H2S is present. We can do this with a list comprehension as below

```
>>> all_points = [i for i in range(len(q_i)) if q_i[i] > 0.]
```

Now, we can create a model with all of these points. We choose the *isotherm_models.binaryisotherm.BinaryLangmuir* model.

```
>>> h2s_binary = BinaryLangmuir(
...     [hat_f_i[i] for i in all_points],
...     [hat_f_j[i] for i in all_points],
...     [q_i[i] for i in all_points],
...     [T[i] for i in all_points],
...     name='H2S_binary'
... )
```

Similarly, we can find the indices of points where *only* H2S is present (i.e., the unary points for H2S), using the following code

```
>>> unary_points = [i for i in range(len(q_i)) if hat_f_j[i] < 1e-12]
>>> f_i = [hat_f_i[i] for i in unary_points]
```

And we can create a unary model as

```
>>> h2s_unary = LangmuirUnary(  
...     f_i,  
...     [q_i[i] for i in unary_points],  
...     [T[i] for i in unary_points],  
...     name='H2S_unary'  
... )
```

We can undertake a similar procedure for CH4, as below

```
>>> df = pd.read_csv('data_sets/CH4_H2S_MFI_binary_with_fugacity.csv')  
>>> hat_f_i, hat_f_j, q_i, T = df['fugacity CH4 [Pa]'], df['fugacity H2S [Pa]'], df[  
↪ 'Q CH4 [mmol/g]'], df['T [K]']  
>>> all_points = [i for i in range(len(q_i)) if q_i[i] > 0.]  
>>> unary_points = [i for i in range(len(q_i)) if hat_f_j[i] < 1e-12]  
>>> f_i = [hat_f_i[i] for i in unary_points]  
>>> ch4_unary = LangmuirUnary(f_i,  
...     [q_i[i] for i in unary_points],  
...     [T[i] for i in unary_points],  
...     name='CH4_unary'  
... )  
>>> ch4_binary = BinaryLangmuir(  
...     [hat_f_i[i] for i in all_points],  
...     [hat_f_j[i] for i in all_points],  
...     [q_i[i] for i in all_points],  
...     [T[i] for i in all_points],  
...     name='CH4_binary'  
... )
```

3.2 Solution

We solve the ch4_unary model first

```
>>> ch4_unary.solve()
```

and observe that the fit is quite good.

```
>>> ch4_unary.get_R2_pyomo()  
0.9998  
>>> ch4_unary.get_objective()  
0.0007123352658190
```

We can take a look at the final parameters that were obtained

```
>>> ch4_unary.dH_i.display()  
dH_i : Size=1  
Key   : Value  
None  : -20205.7398278234
```

```
>>> ch4_unary.q_mi.display()  
q_mi : Size=1  
Key   : Value  
None  : 2.7226241284913613
```

```
>>> ch4_unary.k_i_inf.display()
k_i_inf : Size=1
      Key : Value
      None : 6.61203298602151e-10
```

Then we can do the same thing with the H2S unary model

```
>>> h2s_unary.solve()
>>> h2s_unary.get_R2_pyomo()
0.998700
>>> h2s_unary.get_objective()
0.0053414
```

Alternatively, we can display results at once

```
>>> h2s_unary.display_results()
R2 : Size=1
      Key : Value
      None : 0.9987002690496689
objective : Size=1, Index=None, Active=True
      Key : Active : Value
      None : True : 0.0053414186202173485
H_i_star : Size=1, Index=None
      Key : Lower : Value : Upper : Fixed : Stale : Domain
      None : None : -10.976064382768586 : None : False : False : Reals
A_i : Size=1, Index=None
      Key : Lower : Value : Upper : Fixed : Stale : Domain
      None : None : -7.365904878303015 : None : False : False : Reals
q_mi_star : Size=1, Index=None
      Key : Lower : Value : Upper : Fixed : Stale : Domain
      None : None : 1.0109486926682547 : None : False : False : Reals
q_mi : Size=1
      Key : Value
      None : 3.1127110247255563
k_i_inf : Size=1
      Key : Value
      None : 1.6091644633767268e-10
dH_i : Size=1
      Key : Value
      None : -31300.464752469943
```

Before solving the binary models, it is useful to have a good initial guess. One option is to initialize the binary variables from the Langmuir combining rule

```
>>> h2s_binary.H_i_star = pyo.value(h2s_unary.H_i_star)
>>> h2s_binary.A_i = pyo.value(h2s_unary.A_i)
>>> h2s_binary.q_mi_star = pyo.value(h2s_unary.q_mi_star)
>>> h2s_binary.A_j = pyo.value(ch4_unary.A_i)
>>> h2s_binary.H_j_star = pyo.value(ch4_unary.H_i_star)
>>> ch4_binary.H_i_star = pyo.value(ch4_unary.H_i_star)
>>> ch4_binary.A_i = pyo.value(ch4_unary.A_i)
>>> ch4_binary.q_mi_star = pyo.value(ch4_unary.q_mi_star)
>>> ch4_binary.A_j = pyo.value(h2s_unary.A_i)
>>> ch4_binary.H_j_star = pyo.value(h2s_unary.H_i_star)
```

And then solve them using the usual syntax

```
>>> h2s_binary.solve()
>>> ch4_binary.solve()
>>> h2s_binary.get_R2_pyomo()
0.9988281256
>>> h2s_binary.get_objective()
0.0186995038
>>> ch4_binary.get_R2_pyomo()
0.999329631
>>> ch4_binary.get_objective()
0.007515807
```

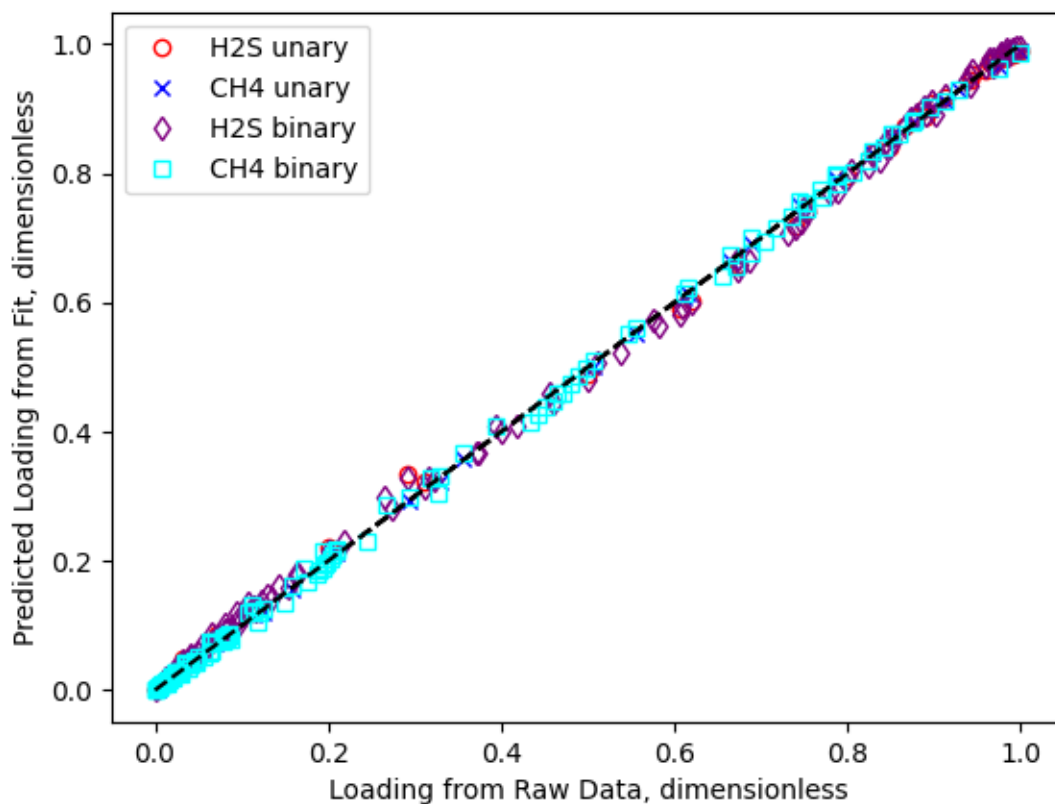
which demonstrates that the fits are again quite good. It is of interest to compare the binary fit parameters to the unary parameters

```
>>> pyo.value(h2s_binary.H_i_star)
-11.073113
>>> pyo.value(h2s_unary.H_i_star)
-10.97606
>>> pyo.value(h2s_binary.q_mi_star)
1.0189875
>>> pyo.value(h2s_unary.q_mi_star)
1.0109486
>>> pyo.value(h2s_binary.A_i)
-7.32572
>>> pyo.value(h2s_unary.A_i)
-7.36590
```

We can also plot all the results to a figure, and save it to a file

```
>>> fig = plt.figure()
>>> fig, ax = h2s_unary.plot_comparison_dimensionless(fig=fig, color='red', marker='o',
↳ markerfacecolor='None', label='H2S unary')
>>> fig, ax = ch4_unary.plot_comparison_dimensionless(fig=fig, ax=ax, color='blue',
↳ marker='x', markerfacecolor='None', label='CH4 unary')
>>> fig, ax = h2s_binary.plot_comparison_dimensionless(fig=fig, ax=ax, color='purple',
↳ marker='d', markerfacecolor='None', label='H2S binary')
>>> fig, ax = ch4_binary.plot_comparison_dimensionless(fig=fig, ax=ax, color='cyan',
↳ marker='s', markerfacecolor='None', label='CH4 binary')
>>> _ = ax.legend()
>>> fig.savefig('docs/source/h2s_ch4_example.png')
```

Which looks like



Having fit the isotherms, we can now evaluate them at arbitrary fugacities and temperatures. We get the same answer whether we use units or dimensional quantities

```
>>> pyo.value(h2s_unary.eval_pyomo(h2s_unary.f_ref, h2s_unary.T_ref) - h2s_unary.q_
↪ref*h2s_unary.eval_dimensionless_pyomo(1., 1.))
0.0
```

```
>>> pyo.value(
... h2s_binary.eval_pyomo(h2s_binary.f_ref, h2s_binary.f_ref, h2s_binary.T_ref)
... - h2s_binary.q_ref*h2s_binary.eval_dimensionless_pyomo(1., 1., 1.)
... )
0.0
```

3.3 Comparison to scipy

```
>>> import numpy as np
>>> popt, pcov = h2s_binary.solve_scipy()
>>> popt
array([ 1.01898639, -11.07321779, -8.15011325, -7.32582652,
       -7.11673647])
>>> popt = np.array(list(map(pyo.value,
... [h2s_binary.q_mi_star, h2s_binary.H_i_star, h2s_binary.H_j_star, h2s_
↪binary.A_i, h2s_binary.A_j])))
```

(continues on next page)

(continued from previous page)

```
array([-1.12501256e-06, -1.04698086e-04, -1.76342848e-04, -1.04982550e-04,  
       -1.81042032e-04])  
>>> h2s_binary.get_R2_scipy()  
0.998828
```

which is nearly the same as the pyomo/ipopt result.

UNARY ISOTHERMS

4.1 Langmuir

The temperature-dependent unary Langmuir isotherm is expressed as

$$q_i = \frac{q_{m,i} k_i f_i}{1 + k_i f_i} \quad (4.1)$$

where f_i , is the fugacity of component i , can be calculated assuming ideal gas

$$f_i^{\text{IG}} = y_i P$$

or, using the [RealGas](#) package to calculate ϕ_i from y_i, P, T data,

$$f_i = \phi_i y_i P$$

An Arrhenius relationship for k_i is assumed as

$$k_i = k_{i,\infty} \exp\left(\frac{-\Delta H_i}{RT}\right)$$

Introducing the dimensionless parameters

$$\theta_i = \frac{q_i}{q_{\text{ref}}} \quad (4.2)$$

$$f_i^* = \frac{f_i}{f_{\text{ref}}} \quad (4.3)$$

$$T^* = \frac{T}{T_{\text{ref}}} \quad (4.4)$$

The variables to be fit in dimensionless form are

$$H_i^* = \frac{\Delta H_i}{RT_{\text{ref}}} \quad (4.5)$$

$$q_{m,i}^* = \frac{q_{m,i}}{q_{\text{ref}}} \quad (4.6)$$

$$A_i = \ln(k_{i,\infty} f_{\text{ref}}) \quad (4.7)$$

So that Equation (4.1) becomes

$$\theta_i = \frac{q_{m,i}^* \exp\left(A_i - \frac{H_i^*}{T^*}\right) f_i^*}{1 + \exp\left(A_i - \frac{H_i^*}{T^*}\right) f_i^*} \quad (4.8)$$

4.2 Modules

class `isotherm_models.unaryisotherm.UnaryIsotherm` (*f_i, q_i, T, q_ref=None, f_ref=None, T_ref=None, **kwargs*)

Base class for Unary Isotherms

Parameters

- **f_i** (*list*) – fugacities of component *i* (can be calculated assuming ideal gas or real gas)
- **q_i** (*list*) – loadings of component *i*
- **T** (*list, optional*) – temperatures in [K], defaults to None
- **q_ref** (*float, optional*) – reference loading, defaults to maximum loading in *q_i*
- **f_ref** (*float, optional*) – reference fugacity, defaults to maximum fugacity in *f_j*
- **T_ref** (*float, optional*) – reference temperature, defaults to maximum temperature in *T*
- **points** (*list, derived from input*) – state points at which a pressure and temperature are provided
- **f_i_star** (*list, derived*) – dimensionless fugacities, calculated by Equation (4.3)
- **theta** (*list, derived*) – dimensionless loadings, calculated by Equation (4.2)
- **T_star** (*list, derived*) – dimensionless temperatures, calculated by Equation (4.4)
- **theta_calc** (*pyo.Var, derived from input*) – calculated dimensionless at each state point
- **objective** (*pyo.Objective, derived from input*) – objective function to be minimized for isotherm fitting, calculated from `isotherm_models.unaryisotherm.UnaryIsotherm.objective_rule_pyomo()`
- **R2** (*pyo.Expression, derived*) – coefficient of determination, see `isotherm_models.unaryisotherm.UnaryIsotherm.R2_rule()`
- **q_calc** (*pyo.Expression, derived*) – calculated loading in units

R2_rule ()

Calculate coefficient of determination squared, R^2

isotherm_eq_rule (*point*)

Constraint for dimensionless expression

objective_rule_pyomo ()

Sum of squared errors between calculated loading and predicted loading

$$\sum_i (\theta_i^{\text{raw}} - \theta_i^{\text{calc}})^2$$

where *raw* denotes the raw data obtained by experiment or molecular simulation and *calc* denotes the data calculated from the isotherm function

solve (*solver=<pyomo.solvers.plugins.solvers.IPOPT.IPOPT object>, **kwargs*)

Solve constraints subject to objective function

Parameters

- **solver** (*pyo.SolverFactory, optional*) – solver for solving model equations, defaults to `pyo.SolverFactory('ipopt')`

- **kwargs** – for solve argument

class isotherm_models.unaryisotherm.LangmuirUnary (*args, **kwargs)

Langmuir isotherm for unary mixture

Isotherm is Equation (4.1). Dimensionless isotherm is Equation (4.8). Dimensionless variables to be fit are H_i^* , A_i , and $q_{m,i}^*$, as defined in Equations (4.5), (4.6), and (4.7), respectively.

Parameters

- **H_i_star** (*pyo.Var*) – H_i^* , dimensionless heat of adsorption of component i
- **A_i** (*pyo.Var*) – A_i , dimensionless langmuir constant in logarithmic space
- **q_mi_star** (*pyo.Var*) – $q_{m,i}^*$, dimensionless saturation loading
- **q_mi** (*pyo.Expression*) – langmuir saturation loading
- **k_i_inf** (*pyo.Expression*) – langmuir adsorption constant independent of temperature
- **dH_i** (*pyo.Expression*) – heat of adsorption of component i

dimensionless_isotherm_expression (*point*)

Dimensionless isotherm expression, see Equation (4.8)

eval ($f_i, T, q_{mi}, k_{i_inf}, dH_i$)

evaluate using generic types (any type)

eval_pyomo (f_i, T)

evaluate using pyomo types (any type)

initial_guess_A_i ()

Initial guess for A_i variable

Todo: Come up with logical initial guess

initial_guess_H_i_star ()

Initial guess for H_i^* variable

This value of 10 corresponds to an absolute value for heat of adsorption of $10RT$ which is approximately 25 kJ/mol

initial_guess_q_mi_star ()

Initial guess for $q_{m,i}^*$ variable

If q_{ref} is chosen to be the saturation loading, $q_{m,i}^*$ will be 1. Thus, we return 1 as initial guess

initial_guess_vector ()

p0 in scipy curve fit; initial guess for *dimensionless* parameters

Note: order here must be the same as last args in `LangmuirUnary.eval_dimensionless()`

isotherm_expression (*point*)

Isotherm expression in unit quantities, see Equation (4.1)

BINARY ISOTHERMS

5.1 Binary Langmuir

$$q_i = \frac{q_{m,i} k_i \hat{f}_i}{1 + k_i \hat{f}_i + k_j \hat{f}_j} \quad (5.1)$$

Arrhenius relationships are used for k_i and k_j , and dimensionless variables are used as illustrated in *isotherm_models.unaryisotherm.LangmuirUnary*

Note: This isotherm is not equivalent to the conventional extended langmuir isotherm, because *both* k_i and k_j are fit simultaneously to binary data.

For completeness, the relationships are repeated for the binary case below

$$\begin{aligned} k_i &= k_{i,\infty} \exp\left(\frac{-\Delta H_i}{RT}\right) \\ k_j &= k_{j,\infty} \exp\left(\frac{-\Delta H_j}{RT}\right) \end{aligned} \quad (5.2)$$

(5.4)

The dimensionless parameters θ_i and T^* are calculated as the unary case, as shown in Equations (4.2) and (4.4), respectively. The other dimensionless parameters are

$$\hat{f}_i^* = \frac{\hat{f}_i}{f_{\text{ref}}} \quad (5.5)$$

$$\hat{f}_j^* = \frac{\hat{f}_j}{f_{\text{ref}}} \quad (5.6)$$

The dimensionless variables to be fit include H_i^* , $q_{m,i}^*$, A_i , H_j^* , $q_{m,j}^*$, and A_j . The former three (H_i^* , $q_{m,i}^*$, and A_i) have the same expression as the unary case, as shown in Equations (4.5), (4.6), and (4.7), respectively. The latter two are expressed as

$$H_j^* = \frac{\Delta H_j}{RT_{\text{ref}}} \quad (5.7)$$

$$A_j = \ln(k_{j,\infty} f_{\text{ref}}) \quad (5.8)$$

So that Equation (5.1) becomes

$$\theta_i = \frac{q_{m,i}^* \exp\left(A_i - \frac{H_i^*}{T^*}\right) \hat{f}_i^*}{1 + \exp\left(A_i - \frac{H_i^*}{T^*}\right) \hat{f}_i^* + \exp\left(A_j - \frac{H_j^*}{T^*}\right) \hat{f}_j^*} \quad (5.9)$$

5.2 Modules

class `isotherm_models.binaryisotherm.BinaryIsotherm`(*hat_f_i*, *hat_f_j*, *q_i*, *T*,
f_ref=None, ***kwargs*)

Base class for Binary Isotherms, inherits from UnaryIsotherm

The following additional dimensionless variables are used in computations

Parameters

- **hat_f_j**(*list*) – mixture fugacities of component *i*
- **hat_f_j** – mixture fugacities of component *j*
- **q_i**(*list*) – loadings of component *i*
- **T**(*list*, *optional*) – temperatures in [K], defaults to None
- **points**(*list*, *derived from input*) – state points at which a pressure and temperature are provided
- **hat_f_i_star**(*list*, *derived*) – dimensionless fugacities of component *i*, calculated by Equation (5.5)
- **hat_f_j_star**(*list*, *derived*) – dimensionless fugacities of component *j*, calculated by Equation (5.6)
- **theta**(*list*, *derived*) – dimensionless loadings, calculated by Equation (4.2)
- **T_star**(*list*, *derived*) – dimensionless temperatures, calculated by Equation (4.4)
- **theta_calc**(*pyo.Var*, *derived from input*) – calculated dimensionless at each state point
- **objective**(*pyo.Objective*, *derived from input*) – objective function to be minimized for isotherm fitting, calculated from `isotherm_models.unaryisotherm.UnaryIsotherm.objective_rule_pyomo()`
- **R2**(*pyo.Expression*, *derived*) – coefficient of determination, see `isotherm_models.unaryisotherm.UnaryIsotherm.R2_rule()`
- **q_calc**(*pyo.Expression*, *derived*) – calculated loading in units
- **unary_points**(*list*, *derived*) – points where only *i* is present, derived from where $\hat{f}_j < 1 \times 10^{-12}$

plot_adsorption_surface()
plot surface of adsorption data

Todo: implement this helpful for debugging

class `isotherm_models.binaryisotherm.BinaryLangmuir`(**args*, ***kwargs*)

Temperature-dependent extended unary Langmuir isotherm, expressed as

Isotherm is Equation (5.1). Dimensionless isotherm is Equation (5.9). Dimensionless variables to be fit are H_i^* , A_i , $q_{m,i}^*$, H_{jstar} , and A_j , as defined in Equations (4.5), (4.6), (4.7), (5.7), and (5.8), respectively.

Parameters

- **H_i_star**(*pyo.Var*) – H_i^* , dimensionless heat of adsorption of component *i*
- **A_i**(*pyo.Var*) – A_i , dimensionless langmuir constant in logarithmic space

- **H_j_star** (*pyo.Var*) – H_j^* , dimensionless heat of adsorption of component j
- **A_j** (*pyo.Var*) – A_j , dimensionless langmuir constant in logarithmic space
- **q_mi_star** (*pyo.Var*) – $q_{m,i}^*$, dimensionless saturation loading
- **q_mi** (*pyo.Expression*) – langmuir saturation loading
- **k_i_inf** (*pyo.Expression*) – langmuir adsorption constant independent of temperature
- **dH_i** (*pyo.Expression*) – heat of adsorption of component i
- **k_j_inf** (*pyo.Expression*) – langmuir adsorption constant independent of temperature
- **dH_j** (*pyo.Expression*) – heat of adsorption of component j

dimensionless_isotherm_expression (*point*)

Dimensionless isotherm expression, see Equation (5.9)

eval ($x, q_{mi}, dH_i, dH_j, k_{i_inf}, k_{j_inf}$)

evaluate using generic types (any type)

eval_pyomo ($\hat{f}_i, \hat{f}_j, T$)

Evaluate isotherm in dimensional form

initial_guess_vector ()

p0 in scipy curve fit; initial guess for *dimensionless* parameters

Note: order here must be the same as last args in `LangmuirUnary.eval_dimensionless()`

isotherm_expression (*point*)

Isotherm expression in unit quantities, see Equation (5.1)

ADSORPTION EQUILIBRIA RULES

For gases and their mixtures, the rules, limits, and consistency tests are [TM88]

1. Unary isotherms should reduce to Henry's law at the limit of zero pressure.
2. In spite of incorrect limits at zero pressure, both the Toth and DR equations are accurate for calculating spreading pressure provided the pressure is sufficiently high.
3. At fixed temperature and pressure, thermodynamically consistent x - y diagrams intersect each other at least once. This can be derived by considering the Gibbs adsorption isotherm at constant spreading pressure.
4. Mixed gas isotherms should display continuity with single-gas isotherms. That is,

$$\lim_{y_i \rightarrow 1} q_t = q_i \text{ (constant } P, T)$$

where $q_t = \sum_i q_i$ is the total loading. Discontinuities generate inaccurate values of adsorbate vapor pressure that lower the quality of calculations of mixed-gas adsorption.

5. Isothermal selectivity curves for different vapor compositions should intersect at the limit of zero pressure.
6. Activity coefficients in the adsorbed phase are functions of spreading pressure as well as composition.
7. Imperfections in the gas phase led to corrections in fugacity that are small compared to the effect of nonidealities in the adsorbed phase. In most cases, vapor-phase imperfections may be ignored unless the pressure is above 500 kPa and experimental error is less than a few percent.

REAL ADSORBED SOLUTION THEORY

Todo: implement this into the code

The *Gibbs adsorption isotherm* is

$$-a d\Pi + \sum_i x_i d\mu_i = 0 \quad (7.1)$$

where a is the surface area per mole of adsorbate, Π is the spreading pressure, x_i is the adsorbed mole fraction of component i , and μ_i is the adsorbed-phase chemical potential of component i .

For change in equilibrium conditions,

$$d\mu_i = d\mu_i^g = RT d \ln \hat{f}_i^g \quad (7.2)$$

where $\hat{f}_i^g$ is the fugacity of component i in the gas phase. And the substituting surface area of the adsorbent is

$$A = a \sum_i q_i \quad (7.3)$$

where q_i is the loading of component i . Substituting Equations (7.2) and (7.3) into Equation (7.1) yields

$$\frac{A}{RT} d\Pi = \sum_i q_i d \ln \hat{f}_i^g \quad (7.4)$$

If we have a good description of the multicomponent isotherms,

$$q_i = F(\{\hat{f}_k\})$$

where F is an isotherm function, Equation (7.4) can be simplified to

$$\frac{A\Pi}{RT} = \sum_i \int_0^{\hat{f}_i^g} \frac{q_i}{f'_i} df'_i$$

where f'_i is a dummy variable for integration.

DATA SETS

8.1 CO₂ and N₂ on BEA

Experiment from [PXSL14]

Table 1: N₂ adsorption on BEA, scanned from images in paper

P [atm]	Q [mmol/g]	T [K]	adsorbate
0.5254701189156284	0.07017543859649145	303.	N ₂
0.5923867910171959	0.0789473684210531	303.	N ₂
0.6568240613926388	0.08333333333333348	303.	N ₂
0.7225015761212199	0.09210526315789469	303.	N ₂
0.788179090849801	0.10087719298245634	303.	N ₂
0.8526163612252438	0.10526315789473673	303.	N ₂
0.9195330333268112	0.11403508771929838	303.	N ₂
0.9988401921780905	0.1184210526315792	303.	N ₂
0.41270462401356545	0.06140350877192979	303.	N ₂
0.47590273701602204	0.06578947368421062	303.	N ₂

Table 2: CO₂ adsorption on BEA, scanned from images in paper

P [atm]	Q [mmol/g]	T [K]	adsorbate
---------	------------	-------	-----------

continues on next page

Table 2 – continued from previous page

0.02981369160199132	0.09649122807017552	273.	CO2
0.055855562077436444	0.17543859649122817	273.	CO2
0.10917737342116132	0.3289473684210531	273.	CO2
0.24682058305615331	0.7192982456140355	273.	CO2
0.3137807343637904	0.9035087719298249	273.	CO2
0.3832192004174003	1.0877192982456143	273.	CO2
0.45017500380443043	1.254385964912281	273.	CO2
0.5171318941716123	1.4254385964912284	273.	CO2
0.5828463662253526	1.5833333333333337	273.	CO2
0.6497989086719275	1.7368421052631582	273.	CO2
0.6683960520880888	1.7763157894736845	273.	CO2
0.6894736842105261	1.8245614035087723	273.	CO2
0.7328670188482358	1.9166666666666672	273.	CO2
0.8196504271831995	2.0877192982456148	273.	CO2
0.9113904650101088	2.2587719298245617	273.	CO2
0.9994075958173001	2.4078947368421058	273.	CO2
0.15004021826561445	0.21052631578947345	303.	CO2

continues on next page

Table 2 – continued from previous page

0.21573729863692678	0.29824561403508776	303.	CO2
0.2801941346551011	0.38157894736842124	303.	CO2
0.34464988369312377	0.4605263157894739	303.	CO2
0.4103447901041326	0.5394736842105265	303.	CO2
0.4785190982412659	0.6228070175438596	303.	CO2
0.5429759342594404	0.7061403508771931	303.	CO2
0.6086686667101457	0.7763157894736845	303.	CO2
0.6731244157481683	0.8552631578947372	303.	CO2
0.7375779908258874	0.9254385964912284	303.	CO2
0.8020326528837582	1.0000000000000002	303.	CO2
0.8677253853344638	1.0701754385964914	303.	CO2
0.9321789604121826	1.1403508771929827	303.	CO2
0.9656438183438771	1.1710526315789478	303.	CO2
0.9991108502358745	1.210526315789474	303.	CO2
0.08434205091415034	0.1184210526315792	303.	CO2
0.13265397073849425	0.05701754385964941	343.	CO2
0.10539142155264239	0.052631578947368585	343.	CO2

continues on next page

Table 2 – continued from previous page

0.0595360768712363	0.02631578947368407	343.	CO2
0.20948933672471134	0.08771929824561386	343.	CO2
0.26154155525120115	0.1184210526315792	343.	CO2
0.3297017326463618	0.14473684210526327	343.	CO2
0.39414552490271526	0.17543859649122817	343.	CO2
0.4610676319050413	0.20614035087719307	343.	CO2
0.5267484075740776	0.2280701754385963	343.	CO2
0.5911911128502791	0.2543859649122808	343.	CO2
0.6581132198526054	0.2850877192982457	343.	CO2
0.7225548381486553	0.3070175438596494	343.	CO2
0.7869986304050085	0.3377192982456143	343.	CO2
0.8539196504271831	0.3640350877192984	343.	CO2
0.9196004260962194	0.38596491228070207	343.	CO2
0.9989141068284091	0.4166666666666665	343.	CO2

8.2 H2S and CH4 on MFI

Molecular simulation from [STS15],

Table 3: H2S adsorption on MFI, taken from tables in SI

P [bar]	Q [mmol/g]	d Q [mmol/g]	adsorbate	T [K]
0.001	0.01	0.0003	H2S	298
0.01	0.098	0.001	H2S	298
0.1	0.90	0.01	H2S	298
0.3	1.91	0.01	H2S	298
0.5	2.28	0.01	H2S	298
1.	2.61	0.01	H2S	298
1.5	2.742	0.004	H2S	298
2	2.810	0.005	H2S	298
3	2.90	0.01	H2S	298
4	2.95	0.01	H2S	298
5	2.983	0.004	H2S	298
6	3.010	0.003	H2S	298
7	3.030	0.004	H2S	298
8	3.050	0.004	H2S	298
9	3.064	0.004	H2S	298
10	3.079	0.003	H2S	298
0.001	0.00226	0.00002	H2S	343
0.01	0.0226	0.0002	H2S	343
0.1	0.221	0.003	H2S	343
0.3	0.62	0.01	H2S	343
0.5	0.96	0.01	H2S	343
1.	1.54	0.01	H2S	343
1.5	1.87	0.01	H2S	343
2	2.074	0.004	H2S	343
3	2.321	0.004	H2S	343
4	2.46	0.01	H2S	343
5	2.55	0.01	H2S	343
6	2.62	0.01	H2S	343
7	2.665	0.003	H2S	343
8	2.704	0.004	H2S	343
9	2.737	0.004	H2S	343
10	2.762	0.003	H2S	343
20	2.905	0.004	H2S	343
30	2.969	0.002	H2S	343
40	3.005	0.002	H2S	343
50	3.030	0.003	H2S	343

Table 4: CH4 adsorption on MFI, taken from tables in SI

P [bar]	Q [mmol/g]	d Q [mmol/g]	adsorbate	T [K]
---------	------------	--------------	-----------	-------

continues on next page

Table 4 – continued from previous page

0.001	0.01	0.0003	H2S	298
0.01	0.098	0.001	H2S	298
0.1	0.90	0.01	H2S	298
0.3	1.91	0.01	H2S	298
0.5	2.28	0.01	H2S	298
1.	2.61	0.01	H2S	298
1.5	2.742	0.004	H2S	298
2	2.810	0.005	H2S	298
3	2.90	0.01	H2S	298
4	2.95	0.01	H2S	298
5	2.983	0.004	H2S	298
6	3.010	0.003	H2S	298
7	3.030	0.004	H2S	298
8	3.050	0.004	H2S	298
9	3.064	0.004	H2S	298
10	3.079	0.003	H2S	298
0.001	0.00226	0.00002	H2S	343
0.01	0.0226	0.0002	H2S	343
0.1	0.221	0.003	H2S	343
0.3	0.62	0.01	H2S	343
0.5	0.96	0.01	H2S	343
1.	1.54	0.01	H2S	343
1.5	1.87	0.01	H2S	343
2	2.074	0.004	H2S	343
3	2.321	0.004	H2S	343
4	2.46	0.01	H2S	343
5	2.55	0.01	H2S	343
6	2.62	0.01	H2S	343
7	2.665	0.003	H2S	343
8	2.704	0.004	H2S	343
9	2.737	0.004	H2S	343
10	2.762	0.003	H2S	343
20	2.905	0.004	H2S	343
30	2.969	0.002	H2S	343
40	3.005	0.002	H2S	343
50	3.030	0.003	H2S	343

Table 5: H2S/CH4 Binary adsorption on MFI, taken from tables in SI

T [K]	P [bar]	y H2S [mol/mol]	dY H2S [mol/mol]	Q H2S [mmol/g]	dQ H2S [mmol/g]	Q CH4 [mmol/g]	dQ CH4 [mmol/g]	dH H2S [kJ/mol]	d dH H2S [kJ/mol]	dH CH4 [kJ/mol]	d dH CH4 [kJ/mol]
298	1	0.00472	0.00002	0.0406	0.0001	0.499	0.001	-28.5	0.2	-19.21	0.05
298	1	0.00944	0.00004	0.0816	0.0003	0.492	0.001	-28.5	0.2	-19.22	0.02
298	1	0.0141	0.0001	0.1231	0.0005	0.483	0.001	-28.6	0.1	-19.34	0.03
298	1	0.019	0.0001	0.164	0.001	0.476	0.001	-28.6	0.1	-19.37	0.05

continues on next page

Table 5 – continued from previous page

298	1	0.0239	0.0001	0.204	0.001	0.47	0.001	-28.5	0.1	-19.39	0.03
298	1	0.0288	0.0002	0.246	0.001	0.462	0.001	-28.6	0.1	-19.4	0.1
298	1	0.0337	0.0002	0.287	0.001	0.455	0.001	-28.7	0.1	-19.47	0.04
298	1	0.0385	0.0001	0.33	0.001	0.4473	0.0004	-28.7	0.1	-19.53	0.04
298	1	0.1038	0.0002	0.816	0.001	0.358	0.001	-29.4	0.1	-19.88	0.04
298	1	0.1665	0.0003	1.212	0.002	0.281	0.001	-30.07	0.03	-20.43	0.05
298	1	0.2421	0.0003	1.571	0.002	0.211	0.001	-30.82	0.05	-21	0.1
298	1	0.3341	0.0003	1.873	0.002	0.149	0.001	-31.4	0.1	-21.4	0.1
298	1	0.4433	0.0002	2.112	0.001	0.1	0.0002	-32	0.1	-21.9	0.1
298	1	0.5673	0.0001	2.295	0.001	0.0623	0.0002	-32.5	0.1	-22.2	0.2
298	1	0.7038	0.0001	2.429	0.001	0.0353	0.0002	-32.8	0.1	-22.4	0.2
298	1	0.84866	0.00002	2.534	0.001	0.0151	0.0001	-33	0.1	-22.7	0.3
298	10	0.00167	0.00002	0.0633	0.0001	1.834	0.002	-29.5	0.3	-20.19	0.03
298	10	0.00339	0.00002	0.1265	0.0001	1.796	0.002	-29.8	0.1	-20.26	0.04
298	10	0.00509	0.00001	0.1898	0.0001	1.758	0.002	-29.9	0.1	-20.35	0.03
298	10	0.0069	0.0001	0.2525	0.0003	1.717	0.002	-29.8	0.1	-20.4	0.1
298	10	0.0088	0.0001	0.315	0.0004	1.682	0.001	-30	0.1	-20.48	0.05
298	10	0.0106	0.0001	0.3784	0.0005	1.643	0.001	-30.3	0.2	-20.46	0.05
298	10	0.0128	0.0001	0.439	0.001	1.605	0.001	-29.9	0.1	-20.49	0.05
298	10	0.0147	0.0001	0.502	0.001	1.567	0.001	-30.2	0.1	-20.54	0.05
298	10	0.043	0.0002	1.235	0.001	1.11	0.001	-31.3	0.1	-21.3	0.1
298	10	0.0807	0.0001	1.789	0.001	0.758	0.001	-32.1	0.1	-21.9	0.1
298	10	0.1415	0.0003	2.251	0.001	0.465	0.001	-32.8	0.1	-22.3	0.1
298	10	0.2364	0.0003	2.575	0.001	0.265	0.001	-33.2	0.1	-22.7	0.1
298	10	0.3628	0.0002	2.773	0.001	0.15	0.001	-33	0.1	-22.4	0.1
298	10	0.5091	0.0001	2.896	0.001	0.0844	0.0004	-33.1	0.1	-22.7	0.1
298	10	0.6667	0.0001	2.977	0.001	0.0443	0.0004	-32.8	0.1	-22.2	0.2
298	10	0.83109	0.00004	3.035	0.001	0.018	0.0001	-32.9	0.1	-22.5	0.4
343	1	0.0079	0.00001	0.0169	0.0001	0.2019	0.0004	-27.8	0.2	-19.15	0.05
343	1	0.01582	0.00003	0.0338	0.0002	0.1998	0.0003	-27.9	0.1	-19.14	0.03
343	1	0.02383	0.00002	0.0504	0.0001	0.1978	0.0002	-28	0.1	-19.14	0.05
343	1	0.03177	0.00003	0.0677	0.0002	0.1955	0.0002	-28	0.1	-19.22	0.03
343	1	0.03976	0.00004	0.0849	0.0003	0.1926	0.0004	-28.1	0.1	-19.2	0.1
343	1	0.0479	0.0001	0.101	0.0004	0.1902	0.0004	-28.1	0.1	-19.2	0.1
343	1	0.0561	0.0001	0.1176	0.0004	0.1872	0.0003	-27.9	0.05	-19.2	0.1
343	1	0.064	0.0001	0.136	0.001	0.1853	0.0004	-28.1	0.1	-19.34	0.05
343	1	0.1643	0.0002	0.339	0.001	0.1577	0.0001	-28.3	0.1	-19.43	0.04
343	1	0.2519	0.0001	0.509	0.001	0.1346	0.0003	-28.47	0.04	-19.57	0.04
343	1	0.344	0.0001	0.675	0.001	0.112	0.0002	-28.75	0.05	-19.78	0.04
343	1	0.4403	0.0001	0.84	0.001	0.09	0.0002	-28.9	0.1	-19.9	0.1
343	1	0.542	0	0.995	0.001	0.0695	0.0001	-29.23	0.04	-20.11	0.03
343	1	0.6487	0.0001	1.146	0.001	0.0498	0.0001	-29.46	0.03	-20.2	0.1
343	1	0.76069	0.00004	1.285	0.001	0.0316	0.00005	-29.74	0.02	-20.6	0.1
343	1	0.87781	0.00002	1.418	0.001	0.01502	0.00005	-29.96	0.02	-20.6	0.2
343	10	0.00368	0.00002	0.0502	0.0001	1.191	0.001	-29	0.1	-19.84	0.04
343	10	0.00743	0.00004	0.1002	0.0003	1.169	0.001	-28.8	0.2	-19.85	0.02
343	10	0.01121	0.00004	0.1502	0.0003	1.146	0.001	-28.8	0.2	-19.91	0.05
343	10	0.0151	0.0001	0.2	0.001	1.124	0.002	-29	0.2	-19.92	0.04
343	10	0.0192	0.0001	0.248	0.001	1.102	0.001	-28.9	0.1	-20.01	0.03
343	10	0.0231	0.0001	0.298	0.001	1.079	0.001	-29	0.1	-20.06	0.04

continues on next page

Table 5 – continued from previous page

343	10	0.027	0	0.3483	0.0002	1.056	0.002	-29	0.1	-20	0.1
343	10	0.0313	0.0001	0.396	0.001	1.036	0.002	-29.2	0.1	-20.05	0.02
343	10	0.0867	0.0005	0.97	0.003	0.78	0.001	-30	0.1	-20.8	0.1
343	10	0.1463	0.0004	1.4	0.003	0.586	0.001	-30.8	0.1	-21.2	0.1
343	10	0.2215	0.0003	1.774	0.001	0.419	0.001	-31.4	0.1	-21.8	0.1
343	10	0.3161	0.0002	2.074	0.001	0.288	0.001	-31.9	0.1	-22.09	0.05
343	10	0.4289	0.0002	2.304	0.001	0.187	0.001	-32.3	0.04	-22.5	0.1
343	10	0.5576	0.0002	2.472	0.001	0.1171	0.0004	-32.5	0.1	-22.7	0.1
343	10	0.6977	0.0001	2.595	0.001	0.0658	0.0004	-32.8	0.1	-22.7	0.2
343	10	0.84586	0.00004	2.69	0.001	0.028	0.0001	-33	0.1	-22.9	0.2
343	50	0.00243	0.00002	0.0599	0.0001	2.097	0.001	-28.5	0.1	-20.06	0.04
343	50	0.00488	0.00003	0.1198	0.0002	2.055	0.001	-28.7	0.2	-20.2	0.02
343	50	0.0075	0.0001	0.1789	0.0004	2.012	0.001	-29	0.2	-20.17	0.05
343	50	0.0101	0.0001	0.2385	0.0003	1.968	0.001	-29.1	0.2	-20.16	0.04
343	50	0.0126	0.0001	0.2979	0.0003	1.925	0.001	-28.9	0.1	-20.21	0.03
343	50	0.0154	0.0001	0.357	0.001	1.882	0.001	-29.2	0.1	-20.25	0.04
343	50	0.0183	0.0001	0.414	0.001	1.84	0.001	-29.2	0.1	-20.3	0.1
343	50	0.021	0.0002	0.474	0.001	1.798	0.001	-29.4	0.1	-20.37	0.02
343	50	0.0614	0.0004	1.152	0.002	1.306	0.002	-29.8	0.1	-20.8	0.1
343	50	0.1101	0.0002	1.657	0.001	0.941	0.001	-30.3	0.1	-21.4	0.1
343	50	0.1794	0.0002	2.081	0.001	0.637	0.001	-31	0.1	-21.8	0.1
343	50	0.2746	0.0002	2.4	0.001	0.412	0.001	-31.2	0.1	-21.96	0.05
343	50	0.3939	0.0002	2.624	0.001	0.259	0.001	-30.83	0.04	-22	0.1
343	50	0.5315	0.0003	2.776	0.002	0.157	0.001	-30.9	0.1	-22.3	0.1
343	50	0.6808	0.0001	2.885	0.001	0.0887	0.0004	-30.5	0.1	-22.1	0.2
343	50	0.8379	0.0001	2.966	0.001	0.0387	0.0002	-30.3	0.1	-23.1	0.2
298	0.001	0	0			0.0006	0.00001				
298	0.01	0	0			0.006	0.0001				
298	0.1	0	0			0.058	0.001				
298	0.3	0	0			0.168	0.003				
298	0.5	0	0			0.274	0.005				
298	1	0	0			0.5	0.01				
298	1.5	0	0			0.7	0.01				
298	2	0	0			0.85	0.01				
298	5	0	0			1.46	0.01				
298	10	0	0			1.88	0.01				
298	12	0	0			1.98	0.01				
298	15	0	0			2.09	0.01				
298	18	0	0			2.18	0.01				
298	20	0	0			2.22	0.01				
298	25	0	0			2.33	0.01				
298	30	0	0			2.39	0.01				
343	0.001	0	0			0.000217	0.000001				
343	0.01	0	0			0.00218	0.000001				
343	0.1	0	0			0.0217	0.0001				
343	0.3	0	0			0.0639	0.0003				
343	0.5	0	0			0.106	0.001				
343	1	0	0			0.204	0.001				
343	1.5	0	0			0.297	0.001				
343	2	0	0			0.379	0.001				

continues on next page

Table 5 – continued from previous page

343	5	0	0			0.788	0.004				
343	10	0	0			1.213	0.003				
343	12	0	0			1.327	0.003				
343	15	0	0			1.472	0.003				
343	18	0	0			1.585	0.002				
343	20	0	0			1.643	0.003				
343	25	0	0			1.779	0.002				
343	30	0	0			1.877	0.003				
343	40	0	0			2.032	0.002				
343	50	0	0			2.141	0.002				
298	0.001	1	0	0.01	0.0003						
298	0.01	1	0	0.098	0.001						
298	0.1	1	0	0.9	0.01						
298	0.3	1	0	1.91	0.01						
298	0.5	1	0	2.28	0.01						
298	1	1	0	2.61	0.01						
298	1.5	1	0	2.742	0.004						
298	2	1	0	2.81	0.005						
298	3	1	0	2.9	0.01						
298	4	1	0	2.95	0.01						
298	5	1	0	2.983	0.004						
298	6	1	0	3.01	0.003						
298	7	1	0	3.03	0.004						
298	8	1	0	3.05	0.004						
298	9	1	0	3.064	0.004						
298	10	1	0	3.079	0.003						
343	0.001	1	0	0.00226	0.00002						
343	0.01	1	0	0.0226	0.0002						
343	0.1	1	0	0.221	0.003						
343	0.3	1	0	0.62	0.01						
343	0.5	1	0	0.96	0.01						
343	1	1	0	1.54	0.01						
343	1.5	1	0	1.87	0.01						
343	2	1	0	2.074	0.004						
343	3	1	0	2.321	0.004						
343	4	1	0	2.46	0.01						
343	5	1	0	2.55	0.01						
343	6	1	0	2.62	0.01						
343	7	1	0	2.665	0.003						
343	8	1	0	2.704	0.004						
343	9	1	0	2.737	0.004						
343	10	1	0	2.762	0.003						
343	20	1	0	2.905	0.004						
343	30	1	0	2.969	0.002						
343	40	1	0	3.005	0.002						
343	50	1	0	3.03	0.003						

continues on next page

Table 6 – continued from previous page

343	50.0	0.0154	0.0001	0.3570	0.0000	0.0000	0.0000	-	0.1	-	0.04	468297	577555	589791	340344	496873	321317	072
343	50.0	0.0183	0.0001	0.414	0.001	1.84	0.001	-	0.1	-	0.1	466915	568185	208358	337463	369176	5399	435
343	50.0	0.021	0.0002	0.474	0.001	1.798	0.001	-	0.1	-	0.02	465628	582365	639243	337455	588798	838	7947
343	50.0	0.0614	0.0004	1.1520	0.0002	0.0000	0.0000	-	0.1	-	0.1	446331	182985	970650	858748	356182	009	018
343	50.0	0.1101	0.0002	1.6569	0.0001	0.0000	0.0000	-	0.1	-	0.1	422977	417385	509560	812982	300788	313	028
343	50.0	0.1794	0.0002	2.081	0.001	0.637	0.001	-	0.1	-	0.1	389605	567936	248919	360765	246688	211	5033
343	50.0	0.2746	0.0002	2.4	0.001	0.4120	0.0000	-	0.1	-	0.05	343581	604828	699728	857360	6790	969	102
343	50.0	0.3939	0.0002	2.6239	0.0001	0.0000	0.0000	-	0.04	-	0.1	285809	451761	424531	104270	224288	238	0158
343	50.0	0.5315	0.0003	2.7760	0.0002	0.0000	0.0000	-	0.1	-	0.1	219369	206873	285254	475976	824836	779	559
343	50.0	0.6808	0.0001	2.885	0.001	0.0887	0.0004	-	0.1	-	0.2	147969	266058	582587	876204	636989	152	8205
343	50.0	0.8379	0.0001	2.966	0.001	0.0387	0.0002	-	0.1	-	0.2	741562	528852	203019	248267	840404	102	66977
298	0.001	0.0	0.0			0.0006	1e-05					99.9998	256676	859999	998256	768570		
298	0.01	0.0	0.0			0.006	0.0001					999.982	567822	459009	982567	822450		
298	0.1	0.0	0.0			0.0579	0.0001					9998.2	169189	859978	256918	898507		
298	0.3	0.0	0.0			0.168	0.003					29984.3	160052	203099	577166	840169		
298	0.5	0.0	0.0			0.2739	0.005					49956.4	381636	639261	128763	272725		
298	1.0	0.0	0.0			0.5	0.01					99825.8	285598	889882	58285	598885		
298	1.5	0.0	0.0			0.7	0.01					149608	284947	599738	856631	71651		
298	2.0	0.0	0.0			0.85	0.01					199303	209533	656196	047668	256		
298	5.0	0.0	0.0			1.46	0.01					495660	855447	492232	171089	48644		
298	10.0	0.0	0.0			1.88	0.01					982718	070449	052827	187344	915213		
298	12.0	0.0	0.0			1.98	0.01					117515	818187	097929	298484	892445		
298	15.0	0.0	0.0			2.09	0.01					146128	612580	687629	041720	45708		
298	18.0	0.0	0.0			2.18	0.01					174439	618819	526910	899344	18183		
298	20.0	0.0	0.0			2.22	0.01					193147	702224	012637	361111	120617		
298	25.0	0.0	0.0			2.33	0.01					239338	701348	726672	355173	949042		
298	30.0	0.0	0.0			2.39	0.01					284714	409069	096497	046968	973215		
343	0.001	0.0	0.0			0.0002	1e-06					99.9999	000580	792999	999000	580772		
343	0.01	0.0	0.0			0.0021	1e-06					999.990	005852	699999	990005	852671		
343	0.1	0.0	0.0			0.0217	0.0001					9999.000	630213	991990	006302	13316		
343	0.3	0.0	0.0			0.0639	0.0003					29991.0	065707	538870	021902	51959		
343	0.5	0.0	0.0			0.106	0.001					49975.0	207485	339925	004149	706745		
343	1.0	0.0	0.0			0.204	0.001					99900.1	095265	599001	079526	5506		
343	1.5	0.0	0.0			0.297	0.001					149775	209903	049085	019935	429392		
343	2.0	0.0	0.0			0.379	0.001					199600	601370	092820	031568	952136		
343	5.0	0.0	0.0			0.7879	0.0009					497507	683045	602510	153660	912042		
343	10.0	0.0	0.0			1.213	0.003					990055	508757	619205	555787	576132		

continues on next page

Table 6 – continued from previous page

343	12.0	0.0	0.0			1.327	0.003					1185690.0117024880785931018158
343	15.0	0.0	0.0			1.472	0.003					1477680.0712029834205141481456
343	18.0	0.0	0.0			1.585	0.002					1767908.0237988822712909992235
343	20.0	0.0	0.0			1.643	0.003					1960420.0980581802100490290725
343	25.0	0.0	0.0			1.7790000020000001						2438310.0519529756240607809398
343	30.0	0.0	0.0			1.876999999999998						2911387.0821865704624273955068
343	40.0	0.0	0.0			2.032	0.002					3843246.9608703605117402175766
343	50.0	0.0	0.0			2.141	0.002					4756285.0176919512570235382224
298	0.001	1.0	0.0	0.01	0.0003							0.0 99.999256119649989925611964946
298	0.01	1.0	0.0	0.098	0.001							0.0 999.92561445500979256144550077
298	0.1	1.0	0.0	0.9	0.01							0.0 9992.56393495019992563934951112
298	0.3	1.0	0.0	1.91	0.01							0.0 29933.125167781977708389260582
298	0.5	1.0	0.0	2.28	0.01							0.0 49814.374643582962874928716486
298	1.0	1.0	0.0	2.61	0.01							0.0 99258.876845249525887684524755
298	1.5	1.0	0.0	2.742	0.004							0.0 148335.56633619889037755740742
298	2.0	1.0	0.0	2.81	0.005							0.0 197046.4926510083232463258002
298	3.0	1.0	0.0	2.9	0.01							0.0 293379.2032039779306773446589
298	4.0	1.0	0.0	2.95	0.01							0.0 388273.202662972063830066574301
298	5.0	1.0	0.0	2.983	0.004							0.0 481744.52506792624890501358445
298	6.0	1.0	0.0	3.01	0.003							0.0 573809.04581507563484096917834
298	7.0	1.0	0.0	3.03	0.004							0.0 664482.48313029492606901874506
298	8.0	1.0	0.0	3.05	0.004							0.0 753780.39953080622254994135085
298	9.0	1.0	0.0	3.063999999999996								0.0 841718.203260635242448067373
298	10.0	1.0	0.0	3.078999999999997								0.0 928311.149730672811149731672
343	0.001	1.0	0.0	0.002260000000000003								0.0 99.999518566166999951856616649
				05								
343	0.01	1.0	0.0	0.02260.0002								0.0 999.95185765969999518576596396
343	0.1	1.0	0.0	0.221	0.003							0.0 9995.1868087882995186808788291
343	0.3	1.0	0.0	0.62	0.01							0.0 29956.702125879185567375292702
343	0.5	1.0	0.0	0.96	0.01							0.0 49879.785998019745957199603149
343	1.0	1.0	0.0	1.54	0.01							0.0 99519.722048319951972204831392
343	1.5	1.0	0.0	1.87	0.01							0.0 148920.67315059728044876703815
343	2.0	1.0	0.0	2.074	0.004							0.0 198083.50153049204175076573659
343	3.0	1.0	0.0	2.320999999999997								0.0 295698.22522059856607507384488
343	4.0	1.0	0.0	2.46	0.01							0.0 392370.73578969809268394742285
343	5.0	1.0	0.0	2.55	0.01							0.0 488107.8320703762156641420627
343	6.0	1.0	0.0	2.62	0.01							0.0 582916.2693276995271155462826
343	7.0	1.0	0.0	2.665	0.003							0.0 676802.7595109665861085015662
343	8.0	1.0	0.0	2.703999999999997								0.0 769773.971520796217464400899
343	9.0	1.0	0.0	2.737	0.004							0.0 861836.531464895785961460721086
343	10.0	1.0	0.0	2.762	0.003							0.0 952997.02291632829970229163286
343	20.0	1.0	0.0	2.905	0.004							0.0 1816406.6513749082033256873854
343	30.0	1.0	0.0	2.969	0.002							0.0 2596545.19674886515065582787
343	40.0	1.0	0.0	3.005	0.002							0.0 3299333.1231588278332807896269
343	50.0	1.0	0.0	3.03	0.003							0.0 3930318.3049747860636609948226

REFERENCES

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

BIBLIOGRAPHY

- [PXSL14] T D Pham, R Xiong, S I Sandler, and R F Lobo. Experimental and Computational Studies on the Adsorption of CO₂ and N₂ on Pure Silica Zeolites. *Microporous and Mesoporous Materials*, 185:157–166, 2014. doi:[10.1016/j.micromeso.2013.10.030](https://doi.org/10.1016/j.micromeso.2013.10.030).
- [RWO+07] R L Rowley, W V Wilding, J L Oscarson, Y Yang, N A Zundel, T E Daubert, and R P Danner. DIPPR® Data Compilation of Pure Chemical Properties, Design Institute for Physical Properties. In *Design Institute for Physical Properties of the American Institute of Chemical Engineers*. AIChE, New York, 2007.
- [STS15] M S Shah, M Tsapatsis, and J Ilja Siepmann. Monte Carlo Simulations Probing the Adsorptive Separation of Hydrogen Sulfide/Methane Mixtures Using All-Silica Zeolites. *Langmuir*, 31:12268–12278, 2015. doi:[10.1021/acs.langmuir.5b03015](https://doi.org/10.1021/acs.langmuir.5b03015).
- [TM88] O Talu and A L Myers. Rigorous Thermodynamic Treatment of Gas Adsorption. *AIChE J.*, 34:1887–1893, 1988. doi:[10.1002/aic.690341114](https://doi.org/10.1002/aic.690341114).

INDEX

B

BinaryIsotherm (class in isotherm_expression() (isotherm_models.binaryisotherm.BinaryLangmuir method), 14
 isotherm_models.binaryisotherm), 18

BinaryLangmuir (class in isotherm_expression() (isotherm_models.unaryisotherm.LangmuirUnary method), 19
 isotherm_models.binaryisotherm), 18

D

dimensionless_isotherm_expression() (isotherm_models.binaryisotherm.BinaryLangmuir method), 19

dimensionless_isotherm_expression() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

E

eval() (isotherm_models.binaryisotherm.BinaryLangmuir method), 19

eval() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

eval_pyomo() (isotherm_models.binaryisotherm.BinaryLangmuir method), 19

eval_pyomo() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

I

initial_guess_A_i() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

initial_guess_H_i_star() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

initial_guess_q_mi_star() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

initial_guess_vector() (isotherm_models.binaryisotherm.BinaryLangmuir method), 19

initial_guess_vector() (isotherm_models.unaryisotherm.LangmuirUnary method), 15

isotherm_eq_rule() (isotherm_models.unaryisotherm.UnaryIsotherm method), 14

L

LangmuirUnary (class in isotherm_models.unaryisotherm), 15

O

objective_rule_pyomo() (isotherm_models.unaryisotherm.UnaryIsotherm method), 14

P

plot_adsorption_surface() (isotherm_models.binaryisotherm.BinaryIsotherm method), 18

R

R2_rule() (isotherm_models.unaryisotherm.UnaryIsotherm method), 14

S

solve() (isotherm_models.unaryisotherm.UnaryIsotherm method), 14

U

UnaryIsotherm (class in isotherm_models.unaryisotherm), 14