Systems & Control Transactions

Research Article - Peer Reviewed Conference Proceeding
ESCAPE 35 - European Symposium on Computer Aided Process Engineering
Ghent, Belgium. 6-9 July 2025
Jan F.M. Van Impe, Grégoire Léonard, Satyajeet S. Bhonsale,
Monika E. Polańska, Filip Logist (Eds.)

PSE PRESS

Langmuir.jl: An Efficient and composable Julia Package for Adsorption Thermodynamics

Vinicius Santana[c]*, Andrés Riedemann[a], Pierre J. Walker[b] and Idelfonso Nogueira[c]

[a] Universidad de Concepción, Department of Chemical Engineering, Concepción, Chile
[b] California Institute of Technology, Division of Chemistry and Chemical Engineering, Pasadena, California, USA
[c] Norwegian University of Science and Technology, Department of Chemical Engineering, Trondheim, Norway
* Corresponding Author: vinicius.santana@ntnu.no.

ABSTRACT

Recent advancements in material design have made adsorption a more energy-efficient alternative to traditional thermally driven separation processes. Accurate modelling of adsorption thermodynamics is crucial for designing and operating equilibrium-limited adsorption systems. High-quality open-source packages like PyIAST, PyGAPsare available for processing adsorption data in Python. They provide a robust set of features for processing and analysing isotherms. However, they have no support for automatic differentiation and are not targeted for performance. Langmuir.jl addresses these limitations by leveraging Julia's composable and differentiable programming ecosystem. Langmuir.jl includes tools for processing adsorption thermodynamics data—loading data, fitting isotherms with most often used models, predictive multicomponent adsorption through Ideal Adsorption Solution Theory (IAST) — and, importantly, enabling accurate derivative calculations through Julia's automatic differentiation libraries, without requiring extensive code adjustments. Furthermore, Langmuir.jl achieves two orders of magnitude faster IAST calculations compared to its Python counterpart, PyIAST.

Keywords: Adsorption, Thermodynamics, Differentiable Programming, Open-Source Software

INTRODUCTION

The separation of fluid mixtures is subject of great attention to chemists and chemical engineers. Traditional methods such as distillation rely on the use of heat to partially vaporize and separate mixtures. Recent advancements in material design have made adsorption a more energy-efficient alternative to traditional thermally driven separation processes [1].

Adsorption is a surface phenomenon that involves the transfer of a substance from a bulk phase of a fluid to the surface of a solid whilst desorption consists of the opposite process. To separate mixtures with this principle, the different affinity of substances (adsorbates) to a solid's surface (adsorbent) is used as driving force. Affinity is linked to the equilibrium concentrations between an adsorbent and adsorbate at a specific temperature. This link is achieved with the well-known adsorption isotherms that seeks to describe the equilibrium relation between bulk concentration, adsorbent concentration and temperature of a fluid or mixture.

Isotherms are fundamental in adsorption as they can be used to describe equilibrium concentrations and to estimate derived quantities such as the Henry coefficient and isosteric heat of adsorption [2]. Classic adsorption isotherms include the Langmuir isotherm, initially developed for gas-phase adsorption on activated carbon, and the simple linear Henry's isotherm. The International Union of Pure and Applied Chemistry (IUPAC) categorizes isotherms into six types based on their shape [3]. With advancements in adsorbents and the discovery of new materials, new isotherms continue to be proposed easily surpassing 20 types today.

When designing adsorption-based separation processes, the typical workflow starts with material characterization and single-component adsorption experiments, as multicomponent adsorption data is challenging to obtain. Single-component data is then fitted to isotherms at different temperatures, enabling the estimation of quantities such as isosteric heat, multicomponent adsorption, and Henry's coefficients, for example. Finally, the predictive capacities of the isotherm and derived

quantities are validated through breakthrough experiments [4].

As the number of isotherms and available data grow with advancements in research and automation, manually performing this workflow becomes increasingly complex, error-prone, and time-consuming. Automating the process with software provides a practical and efficient solution. In the past years, efforts have been made to develop tools that assist users in automating the workflow, enabling the exploration of a broader range of models and potentially enhancing predictive accuracy and process design quality.

While software design has been limited to experts in the past, the advent of high-level dynamically typed programming languages such as Python, Julia and more have made software development possible to a broader range of professionals. In this trend, software for adsorption thermodynamics evolved from statically compiled languages to dynamically typed languages.

IAST++ [5], for example, is an open-source C++ GUI-based software that can fit isotherm data to several models and use the Ideal Adsorbed Solution Theory (IAST) to estimate multicomponent isotherm. Ruptura [4] is also a free and open-source C++ software package for the simulation of gas adsorption breakthrough curves, mixture prediction using IAST-like methods, and fitting of isotherm models' adsorption isotherm data. While C++ is known for its speed and efficient memory management, code syntax is less attractive when compared to high-level languages, which limits cross-collaboration, development, and wider adoption.

Recent advancements in high-level programming languages have led to the development of tools like pyIAST [5] and pyGAPS [6] in Python. PyIAST focuses primarily on providing code interface for initializing and fitting single-component isotherms that can be used for multicomponent prediction through IAST.

In contrast, pyGAPS serves as a "General Adsorption Processing Suite" with a wider range of functionalities. It supports manipulating, storing, visualizing, and processing adsorption isotherms. Its extended feature set includes specific surface area calculations, t- and s-plots, model fitting, IAST calculations, and the estimation of isosteric heat of adsorption from isotherm data.

While Python offers a straightforward and user-friendly environment for software development, particularly with its object-oriented syntax, it often falls short when performance and composability are critical. For instance, the calculation of isosteric heat of adsorption can be performed automatically with exact derivatives, avoiding error approximations and tedious, error-prone derivatives, if automatic differentiation is utilized. However, integrating such functionality directly into tools like pyGAPs could require a significant development effort despite its modular nature. The effort could involve a potential port to a different backend such as JAX or manually writing chain rules for complex operations not supported by numpy-compatible AD frameworks such as autograd – derivatives of root-finding solvers and derivatives of numerical integration routines, for example.

Such a lack of composability in Python often restricts many packages to a specific backend, hindering the adoption and composition of emerging methods from different libraries. The Julia programming language is a language mainly targeted to scientific computing that is known for its composability. It is a fully featured language that can achieve C-like performance native code and, therefore, does not rely heavily on external libraries for numerical computations. Furthermore, it has first-class automatic differentiation support [7]. However, time-to-first execution is a known drawback of JIT compilation in the Julia language. Furthermore, it is a much less popular language than Python, with less mature libraries and a smaller community.

Taking full advantage of Julia's capabilities and its vibrant numerical ecosystem, we present Langmuir.jl: An efficient and composable Julia package for adsorption thermodynamics. Langmuir.jl is part of the Clapeyron-Thermo project that includes the Clapeyron.jl - An Open-Source Fluid Thermodynamics Toolkit in Julia [8] and GCIdentifier.jl – a package for automated group contribution (GC) identification [9].

The remainder of this article is organized as follows: in Section 2, the architecture of Langmuir.jl is explained, as are the isotherm model objects, supported models, core isotherm functions, derived properties, and the use of automatic differentiation. In Section 3, we present case studies of the library and a full workflow for working with adsorption isotherms.

Adsorption Thermodynamics

To clarify the code structure, it is helpful to briefly introduce the physical meaning and mathematical description of adsorption thermodynamics. This begins with the concept of single-component isotherms and extends to properties derived from their definition. These properties include the isosteric heat, Henry's coefficient, and multicomponent adsorption modeled using the Ideal Adsorbed Solution Theory (IAST).

Isotherms

Mathematically, the single component adsorption is a smooth, continuous function that defines the loading/solid phase concentration q_i of the component i as a function of bulk concentration c_i or pressure p_i and temperature [4]:

$$q_i = f(c_i, T) \tag{1}$$

To compute the loading, both temperature and bulk concentration must be specified. While other combinations of independent variables are theoretically valid -

such as using solid-phase concentration and temperature as inputs - such cases are less common and involve additional complexity. This approach requires inverting the function f, either analytically or by solving a root-finding problem.

The temperature dependence in the isotherm must account for the fact that adsorption is an exothermic process. As temperature increases, the loading q_i decreases. This dependency is typically modeled using parameters that explicitly incorporate the effect of temperature in the affinity parameter, such as in Van't Hoff-like expressions or other temperature-dependent terms in the isotherm equation. The single-site Langmuir isotherm temperature dependency can be written as, for example:

$$q_i = \frac{MKp_i}{1+Kp_i} \tag{2}$$

$$K = K_0\, exp\left(-\frac{E}{RT}\right) \tag{3}$$

Henry's Coefficient

The Henry coefficient is the slope of the isotherm as the partial pressure tends to zero. Under this condition, adsorbate-adsorbate interactions are negligible, and adsorption is linearly related to the affinity of the adsorbate at a given temperature. Mathematically, it is defined as:

$$H_i = \frac{\partial q_i}{\partial p_i}|p_{i\to 0} \tag{4}$$

Numerically, the Henry coefficient can be determined in one of three ways. The first method involves manually deriving the isotherm and applying the limit $p_i \to 0$, which, while tedious and prone to human and numerical error, would result in an analytical expression. The second approach is automatic differentiation, which provides an accurate result without introducing numerical errors. Lastly, numerical differentiation offers a simpler alternative but can also introduce numerical errors.

Isosteric Heat of Adsorption

The isosteric heat of adsorption is defined as the differential change in energy when an infinitesimal number of molecules are transferred from the bulk to the adsorbent surface at a fixed temperature, pressure and adsorbent surface area. However, a common way of estimating this quantity is through the Clausius-Clapeyron equation which relates the isosteric heat to the temperature dependence of the adsorption isotherm [2] as follows:

$$q_{st,i} = T\left(V_g - V_a\right)\frac{dp_i}{dT}|_{q_i,A} \tag{5}$$

where V_g is the molar volume of the bulk phase, V_a is the molar volume of the adsorbed phase and $q_{st,i}$ is the isosteric heat. If isotherm is explicit in the loading, one can write:

$$q_{st,i} = -T\left(V_g - V_a\right)\frac{\partial q_i/\partial T|_{p_i}}{\partial q_i/\partial p_i|_T} \tag{2}$$

Similarly to the Henry coefficient, derivatives play an important role in correctly estimating the isosteric heat. This is where the Julia language automatic differentiation support is essential.

Reduced Grand Potential

Similarly to vapor-liquid equilibrium and the definition of fugacity, adsorption equilibrium has an important quantity used in IAST calculations: the Reduced Grand Potential. It can be defined as:

$$\psi_i = \int_0^{P_{i,0}} \frac{q_i(T,p_i)}{p_i}\, dp_i \tag{3}$$

where p_i is the partial pressure, T is the temperature, q_i is the loading, ψ_i is the grand potential.

This can be obtained analytically for certain types of isotherms. When analytical expressions are not possible, it must be approximated numerically.

Langmuir.jl Design and Interface

Julia programming adopts a distinct paradigm compared to the object-oriented approach in Python, relying on composite types and multiple dispatch [7]. While a detailed explanation is beyond the scope of this work, in simple terms, multiple dispatch selects the appropriate method or function to execute based on the input types.

The source code of Langmuir.jl follows best practices in Julia programming and is divided into two main components, as illustrated in **Figure 1**.

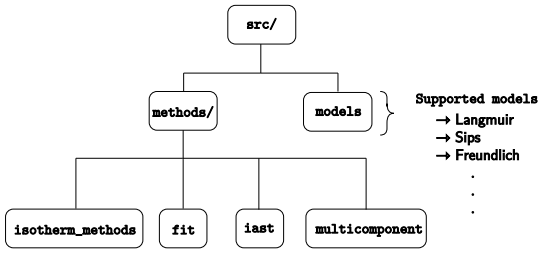

Figure 1. Langmuir.jl source-code structure

The *model* component/folder holds all the files that define supported isotherm models' composite types. Those types have all the parameters needed to define the isotherm completely through their fields. Below is an example of the definition of the Langmuir single-site model:

```
@with_metadata struct LangmuirS1{T} <: Iso-
thermModel{T}
    (M::T, (0.0, Inf), "saturation loading")
    (K₀::T, (0.0, Inf), "affinity parameter")
    (E::T, (-Inf, 0.0), "energy parameter")
End
```

The @with_metadata macro in Julia enhances existing code structures by enabling the addition of extra

inputs that generate additional code. In the example provided, `M` represents the saturation loading, `$K_0$` is the affinity parameter, and `E` is the energy parameter. The annotations, such as parentheses and additional characters beside each field, are processed by the macro to define bounds and provide explanatory information for each parameter. This allows the macro to display these details to the user, improving clarity and usability.

The *method* component is subdivided into 4 subcomponents and files: *fit.jl, iast.jl, isotherm_methods.jl and multicomponent.jl*. *isotherm_methods.jl* holds functions dispatched on isotherms to derive properties of interest, such as the Henry coefficient or the spreading pressure. *fit.jl* holds the functions for fitting isotherms using a global optimization approach. *iast.jl* holds the routines for performing the nested-loop IAST and the FastIAS algorithms [10]. Finally, *multicomponent.jl* has the API for unifying multicomponent adsorption prediction given the single component isotherms and bulk conditions of temperature and partial pressures.

CASE STUDY AND WORKFLOW

This section presents a typical workflow for processing and analyzing isotherm data. The data was obtained from the work of Santana et al. (2024) [11] for an ethane-selective material in an ethane-ethylene mixture, and the results were also compared to what the authors obtained. All the codes for reproducing the results can be found in Langmuir.jl documentation at https://clapeyronthermo.github.io/Langmuir.jl/dev/tutorials/tutorial/.

First, the data is loaded, and then the experimental data is plotted to assess potential isotherms that can fit the data. Then, a model or a set of models is fitted to the data, and the fitting quality is evaluated through a metric and plotted on top of the experimental data. Once the isotherm is obtained, analysis can be performed, such as the isosteric heat of adsorption, Henry coefficient, and multicomponent adsorption estimation with IAST.

Data loading

The data is assumed to be in a standard file format with comma-separated values and loaded into a Julia-native type - an array. The array is then used to construct an isotherm data instance, an abstraction of the Langmuir.jl isotherm. Below is a code snippet. The data can be found at https://github.com/ClapeyronThermo/Langmuir.jl/tree/main/docs/src/tutorials/sample_data.

```
    using Plots, DelimitedFiles, Langmuir
    ethane_data_path = joinpath(@__DIR__,
"sample_data/ethane_tpl_data.csv")
    ethylene_data_path = joinpath(@__DIR__,
"sample_data/ethylene_tpl_data.csv")
    ethane_data = readdlm(ethane_data_path,
',')
    P_ethane = ethane_data[:, 2]*1e5
    T_ethane = ethane_data[:, 1]
    l_ethane = ethane_data[:, 3]
    d_ethane = isotherm_data(P_ethane,
l_ethane, T_ethane)
    Ts_ethane, lp_ethane = split_data_by_tem-
perature(d_ethane)
```

The function split_data_by_temperature function uses the isotherm_data object and returns all the temperatures in an vector and a 3D array holding the loading and pressure for each temperature. By plotting the loading pressure pairs for two of the temperatures for both ethane and ethylene, we can visualize the isotherm in **Figure 2**.

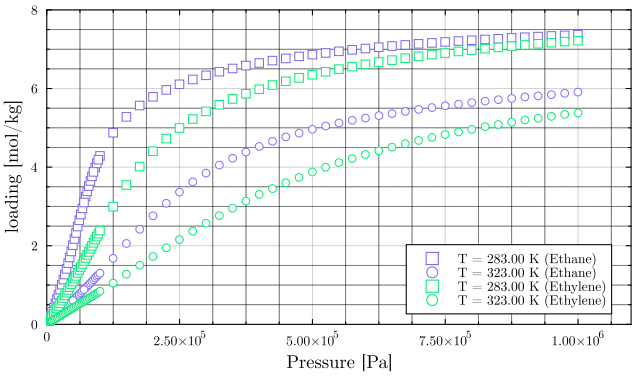

Figure 2. Experimental isotherm data for ethane and ethylene at 283.0 K and 323.0 K.

It can be observed that the isotherms are s-shaped, i.e., the derivative of the loading with respect to the pressure has a maximum. For example, in ethane at 283.00 K, the maximum derivative seems to take place for pressures less than 1.25×10^5 Pa. This behavior is not captured by a single-site Langmuir isotherm. Therefore, a Quadratic isotherm was proposed in the reference manuscript and used in this workflow demonstration.

Isotherm fitting

To fit the quadratic or any other Isotherm in Langmuir.jl, one can first formulate the isotherm fitting problem by providing the isotherm type, the isotherm data wrapped into the isotherm_data object, a fitting criterion, an initial guess and lower and upper bounds for the parameters by using the IsothermFittingProblem abstraction syntax. If initial guesses and lower or upper bouds are not provided, they are defaulted according to each isotherm – more details found in the package's documentation. The code snippet below shows how to formulate the fitting problem for ethane using the mean squared error as the fitting criterion for loading-pressure data, for example.

```
prob_ethane = IsothermFittingProblem(Quadratic{eltype(d_ethane)}, d_ethane, nothing, abs2, x0_ethane, lb_ethane, ub_ethane)
```

Once the fitting problem is set, an optimizer must be selected for minimizing the objective function. In Langmuir.jl differential evolution, which is a global optimization algorithm, is used through the library BlackBoxOptim.jl. Below is a code snippet showing how to set up the optimizer and perform the fitting for ethane isotherm.

```
alg = DEIsothermFittingSolver(max_steps = 5000, population_size = 50,
logspace = true, verbose = true)
loss_fit_ethane, ethane_isotherm = fit(prob_ethane, alg)
```

After running the fitting for both ethane and ethylene and plotting the corresponding models's loadings with experimental values, we obtained the results shown in **Figure 3**.

The loadings for any model can be obtained by calling the function loading as follows:

```
loading(isotherm, p, T)
```

where p is the pressure, T is the temperature and isotherm is an instance of the supported isotherm models.

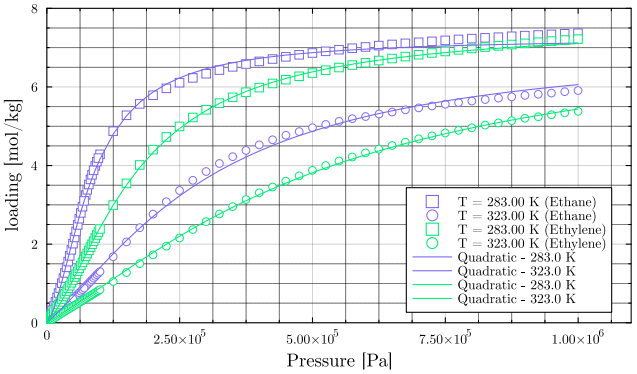

Figure 3. Quadratic isotherm fitting vs experimental data at 283.0 K and 323.0 K.

From the fitting results, the isotherm parameters closely match those reported in the reference manuscript. For example, the saturation loading parameter was estimated at 3.68 in the manuscript, while it is 3.64 in this analysis. However, the energy parameters exhibit a more significant discrepancy as it can be seen from **Table 1** and **Table 2**. This discrepancy is due to a different framing of the optimization problem and different optimization strategies. The reference manuscript imposes the same maximum loading capacity for both components and uses a local optimization method. In our case, no such constraint is set, and we use a global optimizer. Finally, without calorimetric data, resolving the accuracy of these values is challenging, as parameter identifiability is limited and suffers from degeneracy when relying solely on loading data for fitting.

Table 1. Estimated parameters using Langmuir.jl compared to the ones obtained in Santana et al. (2024) [11] for ethane.

	K_A	K_B	E_A (kJ/mol)	E_B (k/mol)	M
Langmuir.jl	2.51×10^{-7}	2.138×10^{-19}	-6.898	-47.789	3.64
Reference	3.214×10^{-8}	1.4237×10^{-14}	-12.19	-48.63	3.68

Table 2. Estimated parameters using Langmuir.jl compared to the ones obtained in Santana et al. (2024) [11] for ethylene.

	K_A	K_B	E_A (kJ/mol)	E_B (k/mol)	M
Langmuir.jl	2.6×10^{-8}	7.13×10^{-19}	-11.79	-41.702	3.83
Reference	6.282×10^{-8}	8.275×10^{-14}	-9.40	-41.70	3.68

Single component isosteric heat of adsorption using automatic differentiation

The isosteric heat can be derived from the isotherm by using Equation 6. In Langmuir.jl, we leverage forward mode automatic differentiation and dual numbers algebra to obtain the derivatives using ForwardDiff.jl. The molar volume of the gas phase is defaulted to the ideal gas molar volume. For the quadratic isotherms, the isosteric heat varies with the amount adsorbed as it can be seen in **Figure 4**. Changes in temperature also affect the isosteric heat but is not depicted in the Figure.

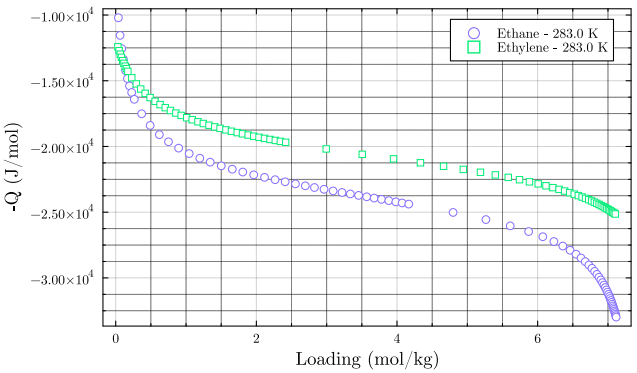

Figure 4. Estimated isosteric heat from isotherm vs loading for ethane and ethylene and 283.0 K.

Multicomponent adsorption with IAST

After obtaining the individual isotherms' parameters for both components, a next step is to use these two isotherms combined to predict the loading in the adsorbent given a mixture of the two components at a certain temperature, pressure and composition. One of the possibles ways to perform such calculations is through the IAST method. To perform IAST calculations in Langmuir, the

two isotherms can first be wrapped into an abstraction named IASTModels where information about the two or more isotherms can be stored and accessed easily:

```
Isotherms = IASTModels(ethane_isotherm, ethylene_isotherm)
```

The loading can be estimated using a similar syntax to the one used for single component isotherm:

```
loading(isotherms, p, T, y)
```

The only difference is that an extra argument (y) must correspond to each component's mole fraction.

Benchmarking IAST – PyIAST and Langmuir.jl

To compare the efficiency of our proposed solution in Julia with existing Python solutions, a benchmarking test was run to solve an IAST problem in PyIAST and Langmuir.jl, considering the isotherms for ethane and ethylene described above. The IAST problem consists of estimating the solid phase composition of ethane and ethylene given a pressure of 101325.0 Pa, temperature of 323K, and equimolar bulk concentration. The isotherms are quadratic parametrized according to **Table 1** and **Table 2** in Langmuir.jl. Since PyIAST does not support temperature dependency in the parameters, isotherms were fitted to both components at 323.0K and used for running IAST. Therefore, the parameters obtained are not the same as those obtained in Langmuir.jl, but they are close.

PyIAST version 1.4.3 using Python 3.11.5 and Langmuir.jl 0.1.1 with Julia 1.10.7 were used. The calculations were run on a laptop with an Intel i5-1250P 1.7 GHz processor and 16 GB of RAM. PyIAST utilizes scipy.root root-finding solver Levenberg-Marquardt (LM) with default tolerances not shown in the documentation. Langmuir.jl has a built-in implementation of nested loop and FastIAS, as described above. Tolerances (relative and absolute) in Langmuir.jl were set to $1x10^{-7}$. To benchmark PyIAST, the Python library time was used, and the same code ran 10000 times. To benchmark Langmuir.jl, the @benchmark macro in BenchmarkTools.jl library was also used with 10000 samples. The mean and standard deviation of run times are displayed on **Table 3**. The Table shows that Langmuir.jl achieves two orders of magnitude faster calculations for this benchmarking problem.

Table 3. Benchmarking of Langmuir.jl and PyIAST (the lower, the better).

Tool / Solver	Mean run time (μs)	Standard Deviation (μs)
PyIAST/LM	206.34	164.35
Langmuir.jl /Nested loop	0.80098	1.5670
Langmuir.jl /FastIAS	0.74962	1.1040

CONCLUSIONS AND FUTURE WORK

This work introduces Langmuir.jl, an open-source Julia library for adsorption thermodynamics. To the best of the author's knowledge, it is the first package in Julia to offer such tools. Langmuir.jl benefits from Julia's modern scientific computing environment, known for its user-friendly syntax, efficiency and robust support for differentiable programming. Automatic differentiation, a core feature of Julia, is particularly advantageous for thermodynamics, where derivatives play a critical role, ensuring maintainability and extensibility. Compared with PyIAST, Langmuir.jl presents two orders of magnitude faster calculations in the IAST problem. However, it is placed in a less popular programming language and has fewer features than PyGAPs, for example. Future developments aim to integrate property calculations, such as loading and isosteric heat, into breakthrough curve simulations, similar to the functionality provided by Ruptura software.

DIGITAL SUPPLEMENTARY MATERIAL

The source code and documentation for Langmuir.jl can be found at https://github.com/Clapeyron-Thermo/Langmuir.jl/tree/main.

ACKNOWLEDGEMENTS

This publication has been produced with support from the HYDROGENi Research Centre (hydrogeni.no), performed under the Norwegian research program FMETEKN. The authors acknowledge the industry partners in HYDROGENi for their contributions and the Research Council of Norway (333118).

REFERENCES

1. J.-R. Li et al., “Carbon dioxide capture-related gas adsorption and separation in metal-organic frameworks,” Coord. Chem. Rev., vol. 255, no. 15, pp. 1791–1823, 2011, doi: https://doi.org/10.1016/j.ccr.2011.02.012.
2. H. Pan, J. a Ritter, and P. B. Balbuena, “from the Clausius - Clapeyron Equation,” vol. 7463, no. 11, pp. 6323–6327, 1998.
3. M. A. Al-Ghouti and D. A. Da’ana, “Guidelines for the use and interpretation of adsorption isotherm models: A review,” J. Hazard. Mater., vol. 393, p. 122383, 2020, doi: https://doi.org/10.1016/j.jhazmat.2020.122383.
4. S. Sharma et al., “RUPTURA: simulation code for breakthrough, ideal adsorption solution theory computations, and fitting of isotherm models,” Mol. Simul., vol. 49, no. 9, pp. 893–953, 2023, doi: 10.1080/08927022.2023.2202757.
5. C. M. Simon, B. Smit, and M. Haranczyk, “pyIAST:

Ideal adsorbed solution theory (IAST) Python package," Comput. Phys. Commun., vol. 200, pp. 364–380, 2016, doi: https://doi.org/10.1016/j.cpc.2015.11.016.
6. P. Iacomi and P. L. Llewellyn, "pyGAPS: a Python-based framework for adsorption isotherm processing and material characterisation," Adsorption, vol. 25, no. 8, pp. 1533–1542, 2019, doi: 10.1007/s10450-019-00168-5.
7. J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," SIAM Rev., vol. 59, no. 1, pp. 65–98, 2017, [Online]. Available: https://doi.org/10.1137/141000671
8. P. J. Walker, H. W. Yew, and A. Riedemann, "Clapeyron.jl: An Extensible, Open-Source Fluid Thermodynamics Toolkit," Ind. Eng. Chem. Res., vol. 61, no. 20, pp. 7130–7153, 2022, doi: 10.1021/acs.iecr.2c00326.
9. P. J. Walker, A. Riedemann, and Z.-G. Wang, "GCIdentifier.jl: A Julia package for identifying molecular fragments from SMILES," J. Open Source Softw., vol. 9, no. 96, p. 6453, 2024, doi: 10.21105/joss.06453.
10. Z. Qiao, Z. Wang, C. Zhang, S. Yuan, Y. Zhu, and J. Wang, "PVAm–PIP/PS composite membrane with high performance for CO2/N2 separation," AIChE J., vol. 59, no. 4, pp. 215–228, 2012, doi: 10.1002/aic.
11. V. V. Santana et al., "Ethylene Purification by Pressure Swing Adsorption with the Paraffin Selective Metal-Organic Framework—DUT-8," Ind. Eng. Chem. Res., vol. 63, no. 5, pp. 2307–2319, 2024, doi: 10.1021/acs.iecr.3c02808.