pyIsoP Documentation

pyisop

Apr 29, 2020

CONTENTS:

PyIsoP uses a fast and accurate, semi-analytical algorithm to calculate the adsorption of single-site molecules in nanoporous materials using energy grids. The energy grid algorithm in PyIsoP is parallelized using Dask (python) such that PyIsoP will work just as well on your laptop (1 CPU, multi-threading) and on a High Performance Cluster (hundreds of CPU, multi-processing) with no or little additional coding on your part. The method itself is about 100 times faster compared to grand canonical Monte Carlo (GCMC) simulations, combined with the fast calculation of energy grids PyIsoP is ideal for obtaining quick estimates of adsorption and even interactive high-throughput screening of large databases. Although originally developed for predicting hydrogen adsorption, the underlying algorithm can be readily applied to other molecules which can also be modeled by a single-site (spherical probe) such as methane and noble gases. Since the energy landscape of a material is usually independent of temperature[1], including thermal swing into our calculations is also quick and easy. The energy grids can also be used to visualize and analyze complex pores in a materials by choosing the right isoenergy contours from the energy landscape. Please refer to our documentation page on ReadTheDocs for theory, examples and the API reference.

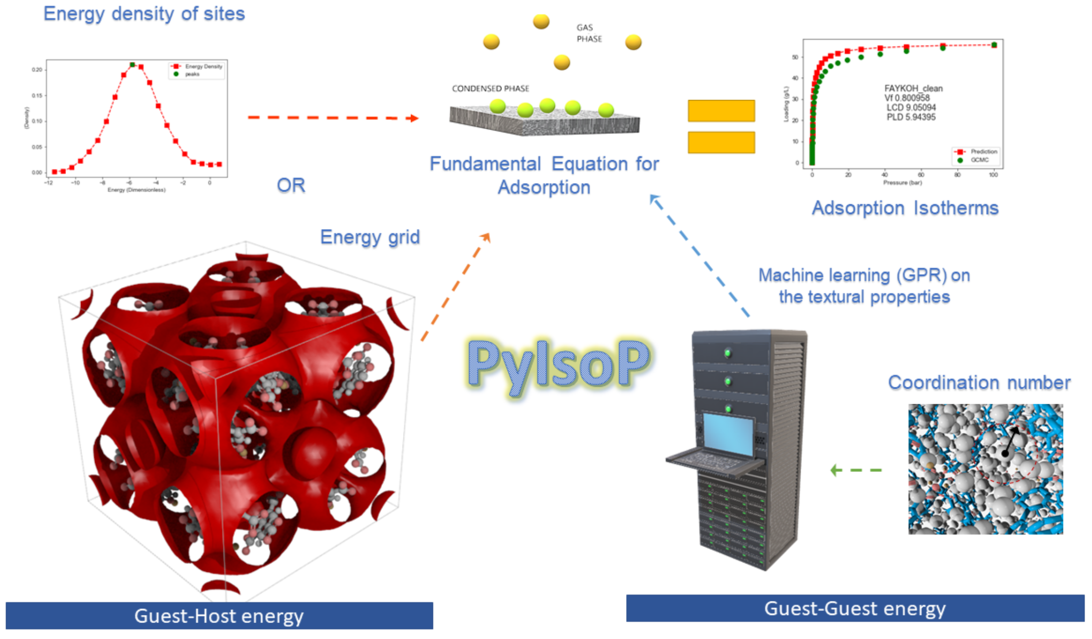
[1] Feynman-Hibbs correction induces a temperature dependency on the energy grid, however this maybe assumed to be weak. For polyatomic probes, the existence of different orientations at any given site also imparts a temperature dependence on the energy grid.

CHAPTER

ONE

HOW DOES IT WORK. . . ?

Although PyIsoP offers many functionalities, the overall approach can be summarized as shown



CHAPTER TWO

COMING SOON !

We are currently working on adding an automated, energy-based, molecular siting module and extending the isotherm prediction approach to ethane and higher alkanes. Stay tuned for new features, tests, bug-fixes and examples.

2.1 Getting Started

2.1.1 Installing PyIsoP

PyIsoP is deployed on PyPI , we can install it easily using pip

```
pip install pyisop
```

or clone from github

```
git clone git@github.com:arung-northwestern/pyIsoP.git
cd pyIsoP/
python setup.py install
```

Dependencies

To use all the functionality of PyIsoP the following python packages are required, which if not present are also installed automatically using pip.

- Python 3.6 or newer
- Numpy 1.13.3 or newer
- Scipy 1.1.0 or newer
- Sklearn 0.19.1 or newer
- ASE 3.16.0 recommended (newer versions have trouble reading RASPA generated PDB files).
- PYEVTK 1.1.1 or newer
- Pandas 0.20.3 or newer
- Numba 0.35 or newer
- PyTest 5.0.1 or newer
- Dask 2.2.0 or newer
- Dask_jobqueue 0.6.2 or newer

- Tqdm

2.2 Theoretical Background

In this section we provide the theoretical background and the derivation for the equations used in PyIsoP. Please refer to the original publication from Gopalan *et al.*, [GBBS19] for more explanation.

2.2.1 The Local Langmuir constant (K_L)

The free energy of adsorption for a spatially distinct site i inside a material is given by Snurr *et al.* [SBT94],

$$exp\left(\frac{-A_i}{k_B T}\right) = \frac{1}{8\pi^2 V_{uc}} \int_i exp(-\frac{U_i}{k_B T}) d^3 r d^3 \phi \tag{2.1}$$

where k$_B$ is the Boltzmann constant, and $d^3 r$ and $d^3 \phi$ represent the integral over the space forming the site and the orientations of the adsorbate molecule respectively. V_{uc} simply serves as a volume scaling factor. For hydrogen represented by a single site probe, adsorbing at a point (cubelet in space with energy E_i) equation (2.1) reduces to,

$$exp(-\beta A_i) = \frac{V_{cubelet}}{V_{uc}} exp(-\beta E_i) \tag{2.2}$$

where $\beta = 1/k_B T$. Now, if there are $n(E_i)$ such sites in a given volume V, the combined Henry's constant resulting from the adsorption in all these sites, each of energy E_i will be

$$K_H(E_i, T) = \frac{M_A}{N_A} \frac{V_{uc}}{V} \frac{1}{k_B T} n(E_i) exp(-\beta A_i) \quad [mass/(pres.vol.)] \tag{2.3}$$

substitute equation (2.2) in equation (2.3) to get

$$K_H(E_i, T) = \frac{M_A}{N_A} \frac{1}{k_B T} . \frac{V_{cubelet}.n(E_i)}{V} exp(-\beta E_i) \tag{2.4}$$

The resultant Langmuir constant from the adsorption in a site with the energy E_i is related to $K_H(E_i)$ as

$$K_L(E_i, T) = \frac{K_H(E_i)}{Q_{sat}(E_i)} \quad [1/pres.] \tag{2.5}$$

where $Q_{sat}(E_i)$ represents the overall saturation capacity of all sites with the energy E_i

$$Q_{sat}(E_i) = Q_{sat} * f_V(E_i) = Q_{sat} * \frac{n(E_i) V_{cubelet}}{V} \tag{2.6}$$

where $f_V(E_i)$ is the fraction of the total void volume that is formed by all sites of energy E_i. Using equations (2.4) and (2.6), equation (2.5) can be simplified to

$$K_L(E_i, T) = \frac{M_A}{RTQ_{sat}} exp(-\beta E_i) \tag{2.7}$$

2.2.2 The Fundamental Equation of adsorption

Adsorption of a pure component in a nanoporous material under a given set of conditions (T,P) can be considered as a summation over all of its adsorption sites. Amongst all the possible ways to characterize an adsorption site, one useful descriptor is its interaction energy with the guest molecule. Now the overall adsorption in the material at any given T,P is the sum over all the energetically different adsorption sites,

$$Q(T, P) = \sum_i Q_{sat} . f_V(E_i) \frac{K_L(E_i, T).P}{K_L(E_i, T).P + 1} \tag{2.8}$$

Now if we assume that there is a continuous distribution of such sites in the material, which we can think in terms of a normalized probability density $\rho(E)$. This normalized density $\rho(E)$ is related to $f_V(E_i)$ (from equation (2.6)), the fraction of the total void volume that is formed by all sites of energy E as,

$$f_V(E) = \rho(E)dE \tag{2.9}$$

where $f_V(E)$ equals the probability of finding a site of energy between E and $E + dE$ (once we assume a continuous distribution) inside the box. Hence the overall adsorption in the material can be expressed as a continuous integral involving the saturation loading Q_{sat}, the local Langmuir constants K_L and the density of sites $\rho(E)$.

$$\frac{Q(T,P)}{Q_{sat}} = \int_{-\infty}^{\infty} \frac{K(T,E) * P}{1 + K(T,E) * P}\rho(E)dE \tag{2.10}$$

To make accurate predictions at high pressures (order of 10 bar), the local Langmuir constant should be corrected for the guest-guest contribution. Including the guest-guest interactions into E creates a pressure dependency in K_L. We include that explicitly to get

$$\frac{Q(T,P)}{Q_{sat}} = \int_{-\infty}^{\infty} \frac{K(T,P,E) * P}{1 + K(T,P,E) * P}\rho(E)dE \tag{2.11}$$

where the energy E now includes both the contributions,

$$E = E_{guest-host}^{(x,y,z)} + E_{guest-guest}^{(T,P,material)} \tag{2.12}$$

Calculation of the guest-host and guest-guest contributions, is explained in sections *guest-host* and *guest-guest* respectively.

2.2.3 The Saturation capacity

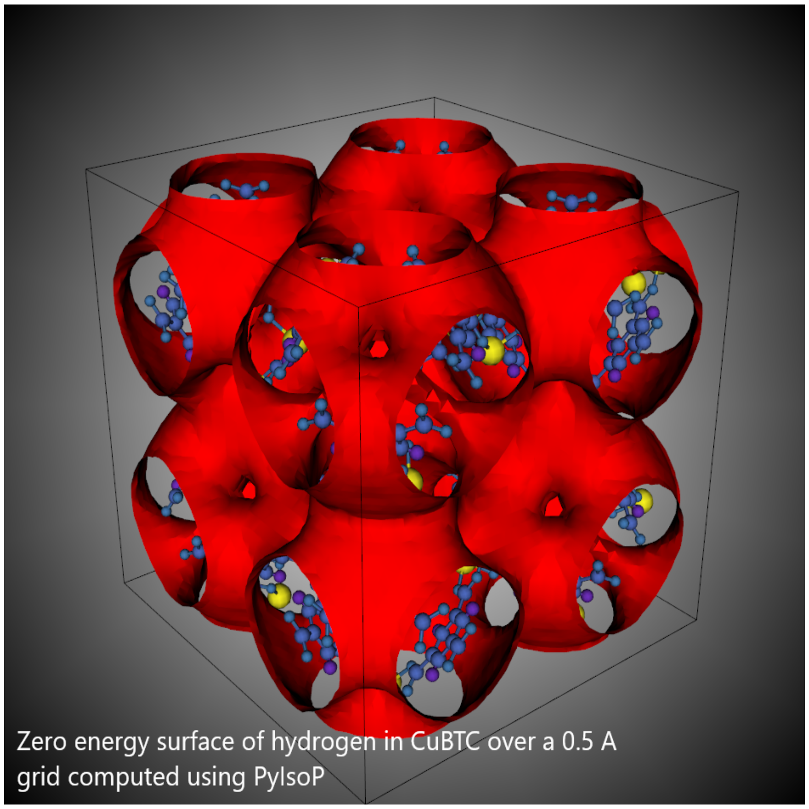
For hydrogen, the saturation loading can approximated as [SKL+14],

$$Q_{sat} = \rho_{L,H_2} * V_f = 70.8(g/L) * V_f$$

where $\rho_{\mathrm{L,H_2}}$ is the liquid density of hydrogen and V_f is the void fraction of the material.

2.2.4 Guest-Host interactions

The guest-host energy can be easily calculated by dividing the material into cubelets and calculating the energy of hydrogen placed at each position using a classical force field. The histogram of this energy grid provides us with the density distribution of sites $\rho(E)$ to use with equation (2.11).


Zero energy surface of hydrogen in CuBTC over a 0.5 A grid computed using PyIsoP

2.2.5 Guest-Guest Interactions

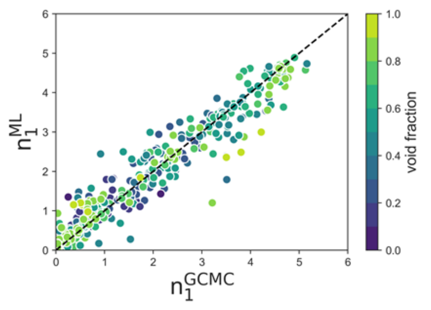
The guest-guest interaction energy of hydrogen inside a nanopore on the other hand depends upon the adsorption conditions (T,P) and the effects of confinement in the material. The difficulty in understanding phase of hydrogen inside the nanopores of a material, under different adsorption conditions, makes theoretical descriptions of the guest-guest interactions almost impossible. One of the most useful descriptors of the local environment of molecules is the average first-shell coordination number (n_1) [Was80]. The coordination number for hydrogen in a material at a given (T,P) can be calculated from GCMC simulation as the integral of the radial distribution function $(g(r))$ of hydrogen inside a nanopore.

$$n_1^{T,P} = \int_{r_0}^{r_1} 4\pi r^2 dr \rho_b g(r) \tag{2.13}$$

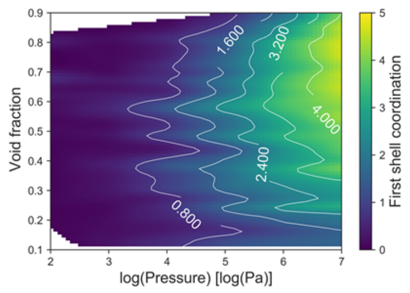
where r_0 and r_1 correspond to the range of the first peak in $g(r)$ and ρ_b is the overll box number density. If we neglect all the interactions beyond the first coordination shell, the average guest-guest energy can then be approximated in terms of the first-shell coordination number and the Lennard Jones well-depth (ϵ) as

$$E_{guest-guest}^{T,P} = \frac{n_1^{T,P} * \epsilon}{2} \tag{2.14}$$

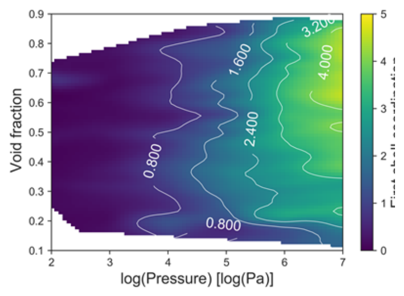
Hence, in order to make predictions with the model we need a means to predict the coordination number of hydrogen in a material at a given (T,P). For example in our recent study [GBBS19], we trained a machine learning model which uses Gaussian Process Regression (GPR) [Ras04] with {log(P), T, void fraction (V_f), the largest cavity diameter (LCD) and the pore limiting diameter (PLD)} as the feature space. The model was trained on data from the GCMC simulations performed on a set of 1000 representative MOFs from the CoRE-MOF (**1.0_**) [CCH+14] The parity between GCMC and GPR and the coordination number landscapes at 77 K are reproduced below. PyISoP offers machine learning with the same settings (GPR, kernel: Rational Quadratic, Length-scale = 0.5) for quick training on your data. To use a different kernel or a machine learning algorithm one could use Scikit-learn [PVG+11] and feed the coordination number array into PyIsoP's predictor to make isotherm predictions.



Parity in coordination numbers GCMC vs GPR



Coordination number landscape from GCMC



Coordination number landscape from GPR

2.2.6 Final Form of K_L

If we put all the pieces together, the final expression of the local Langmuir constant to be used in equation (2.11) looks like

$$K_L(E^{bin}_{guest-host}, T, P) = exp\big(-\beta E^{bin}_{guest-host}\big) \cdot exp(-\beta \frac{n_1^{T,P}\epsilon}{2})$$

2.3 Examples

Here we provide a few simple examples on using the different functionalities offered by PyIsoP. There are more options available to the individual functions than the ones demonstrated here. Please refer to the corresponding API reference to find more about the modules and functions and the possible options.

2.3.1 Energy Grid Calculation

PyIsoP can compute energy grids using the grid calc or the grid_calc_dask (recommended) functions from the grid3D module. The functional forms for the interatomic potentials (Eg. Lennard-Jones) are defined in the potentials module and the force field parameters are read by the forcefields module into an instance. Please take a look at 'pyIsoP\forcefield\UFF\force_field_mixing_rules.def' file on GitHub for a example on specifying LJ parameters for use with PyIsoP. You will see that the formatting is very similar to that of RASPA except for the '_' often used in atom typing.

1. Calculation using Numba (serial)

This is an older implementation that uses vectorized from Numba for computing energy grids quickly on a single CPU (multi-threading). Although orders of magnitude faster than 'for' loops, this implementation lacks in scalability and algorithmic efficiency. It is still quite useful if you do not want to install Dask or if you want the output grid directly as a Numpy array and not as a lazy-evaluated Dask array. However, if you are calculating a fine grid, the resultant array may not fit into your RAM. The Dask option is recommended for such out-of-memory calculations.

```
import pyIsoP.grid3D as grid3D
import pyIsoP.potentials as potentials
import pyIsoP.forcefields as forcefields
import pyIsoP.writer as writer

####################################################################
# Calculate the grid
t1=grid3D.grid3D('ZIF-4_mod.cif',spacing=0.5)          # Intialize grid3D object
f1=forcefields.forcefields(t1, forcefield='C:/PyIsoP/forcefield/UFF', sigma=3.95,␣
↪epsilon=46)       # Update the force field details to grid obj. t1
t1= grid3D.grid3D.grid_calc(t1,"lj",f1)                          # Overwrite the␣
↪existing object with computed 3D grid.

# Save coordinates for visualizing later
writer.writer.write_vts(t1,'zif-4_grid')                   # Write a binary vtk file
writer.writer.write_frame(t1,'zif-4_repeat.pdb')     # Save the corresponding␣
↪replicated structure corresponding to a 12.8 A (default) cut-off.
```

2. Energy Grid Calculation using Dask (serial or paralllel)

PyIsoP uses Dask to compute energy grids in parallel (multi-threading or over many CPUs) such that it will work on your tiny laptop or a High Performance Cluster (HPC) with no or little additional coding on your part. This facilitates super-fast calculation of fine grids and/or even high-throughput screening of materials in an interactive fashion, just as if you were working on your local machine.

We have to initialize a client for Dask depending upon the machine you want to use to compute the grid.

2.1 **Working on your laptop (1 CPU)**: No need to do anything, Dask will automatically recognize your cores and use multi-threading. No overhead will occur from making copies and transfering information between workers.

2.2 **Working on an HPC (many, many CPUs)**: You have to start a mini-cluster with your processor and memory requirements and set that as your client. Here's an example on an HPC that uses SLURM as the job-scheduler

```
from dask_jobqueue import SLURMCluster # My HPC uses SLRURM
from dask.distributed import Client          # Client class

cluster=SLURMCluster(cores=4, interface='ib0', project='p#$###', queue='short
↪', walltime='04:00:00', memory='100GB') # This is one 'job' or worker, it␣
↪has 4 CPUs.
cluster.scale(25)   # We are starting 25 such workers, a total of 100 CPUs
client= Client(cluster) # Use the mini-cluster as the client for our␣
↪calculations.
```

For many other options on setting up Dask, including HPCs, please refer to the Dask setup documentation.

The rest of the code is very similar to that from before, except . . .

```
import pyIsoP.grid3D as grid3D
import pyIsoP.potentials as potentials
import pyIsoP.forcefields as forcefields
import pyIsoP.writer as writer

####################################################################
# Calculate the grid
t1=grid3D.grid3D('ZIF-4_mod.cif',spacing=0.5)          # Intialize grid3D object
```

(continues on next page)

(continued from previous page)

```
f1=forcefields.forcefields(t1, force_field='C:/PyIsoP/forcefield/UFF', sigma=3.95,␣
↪epsilon=46)      # Update the force field details to grid obj. t1
grid_dask = grid3D.grid3D.grid_calc_dask(t1,f1)                           # Returns␣
↪the grid as a Dask array.
```

the energy grid (*grid_dask* in the example above) returned now is a lazy-evaluated Dask array, with all the rules and element formulae embedded within. To evaluate it and append your grid object use

```
# To evaluate it and append your grid object then and there, we can't do anything␣
↪else unless evaluation is complete.
t1.pot = grid_dask.compute()

# To evaluate in the background. We can continue adding rules to the array while the␣
↪array is being 'persisted'.
client.persist(grid_dask) # Starts in the background on an HPC.
grid_dask = grid_dask + 1.2345678 # Continue doing things to the array, just a silly␣
↪example.
```

Please refer to Dask documentation for all the possibilities using Dask-clients and Dask-arrays.

Bonus Example: Interactive, Scalable and High-throughput

Dask-bags are ideal for performing the same function on many items (files, folders etc) in parallel, as long as the order of the bagged items is unimportant. Since Dask is already scalable and interactive, PyIsoP can be readily extended to high-throughput calculation of energy grids (or isotherms) using Dask-bags.

```
####################################################################
# This function computes and returns the energy grid as a Dask array
####################################################################
def compute_grid_pyisop_dask(cif, spacing=0.5):
        from pyIsoP import grid3D, forcefields
        import numpy as np
        t1=grid3D.grid3D(cif)          # Intialize grid3D object
        f1=forcefields.forcefields(t1, force_field='/home/agp971/pyIsoP/forcefield/UFF
↪', sigma=3.95, epsilon=46)      # Update the force field details to grid obj. t1
        grid_dask = grid3D.grid3D.grid_calc_dask(t1,f1)   # Computes the grid as a␣
↪Dask array.
        return grid_dask.compute()                # Returns the grid as a Numpy array␣
↪here itself, only because we are saving space by wrapping things in dask bags later␣
↪anyway.

##############
# Main code
##############
import dask.bag as db # Import the Dask-bags class
import glob    # To dig up a bunch of cif files

cif_list = glob.glob('**/*.cif', recursive=True)                             # Grab␣
↪all the CIF files in the current folder
many_grids=db.from_sequence(np.array(cif_list)).map(compute_grid_pyisop_dask) # Apply␣
↪the function to all the items
many_grids.persist() # start the calculations in the background, returns a future.


# To print all the grids as numpy arrays use
```

(continues on next page)

(continued from previous page)

```
many_grids.compute()

# To print any one (for example the first grid) use.
many_grids.compute()[0]



# Or create a new grid3D object for any of the CIFs (first entry chosen below) and␣
↪save the energy grid there there
t1=grid3D.grid3D(cif_list[0])          # Intialize grid3D object
t1.pot = many_grids.compute()[0]       # Stored the desired grid into the grid3D␣
↪object

# Like before, you can also write this into a binary VTK (.vts) file for␣
↪visualization while specifying the number of unit cells along each direction.
from pyIsoP import writer
writer.writer.write_vts(grid_obj=t1, path_to_file='some_filename', nx_cells=2, ny_
↪cells=2, nz_cells=2)
```

4. Benchmarking on a Fine Energy Grid

Let's try computing the energy grid for Cu-BTC which has a simple-cubic unit cell and an edge length of 26.343 angstroms. Our mini-cluster consists of one or more workers, where each worker (or job) consists of 4 CPUs and a memory of 100 GB (only a small fraction of which will be used here). Since parallel computing involves many stages of information reading, writing and transfer added atop the actual computing time, only the total time is plotted here. We choose a rather fine grid of 0.1 angstrom spacing (**263x263x263=18,191,447 grid points**) and compute it by employing 10 (40 CPUs), 20 (80 CPUs), 25(100 CPUs) and 30 (120 CPUs) workers respectively with each worker having a maximum memory of 100 GB. We see that even at 120 CPUs, the total computation time continues to drop linearly, which indicates that we haven't hit the point of diminishing returns yet, at least for this fine of a grid on this material.

Notes:

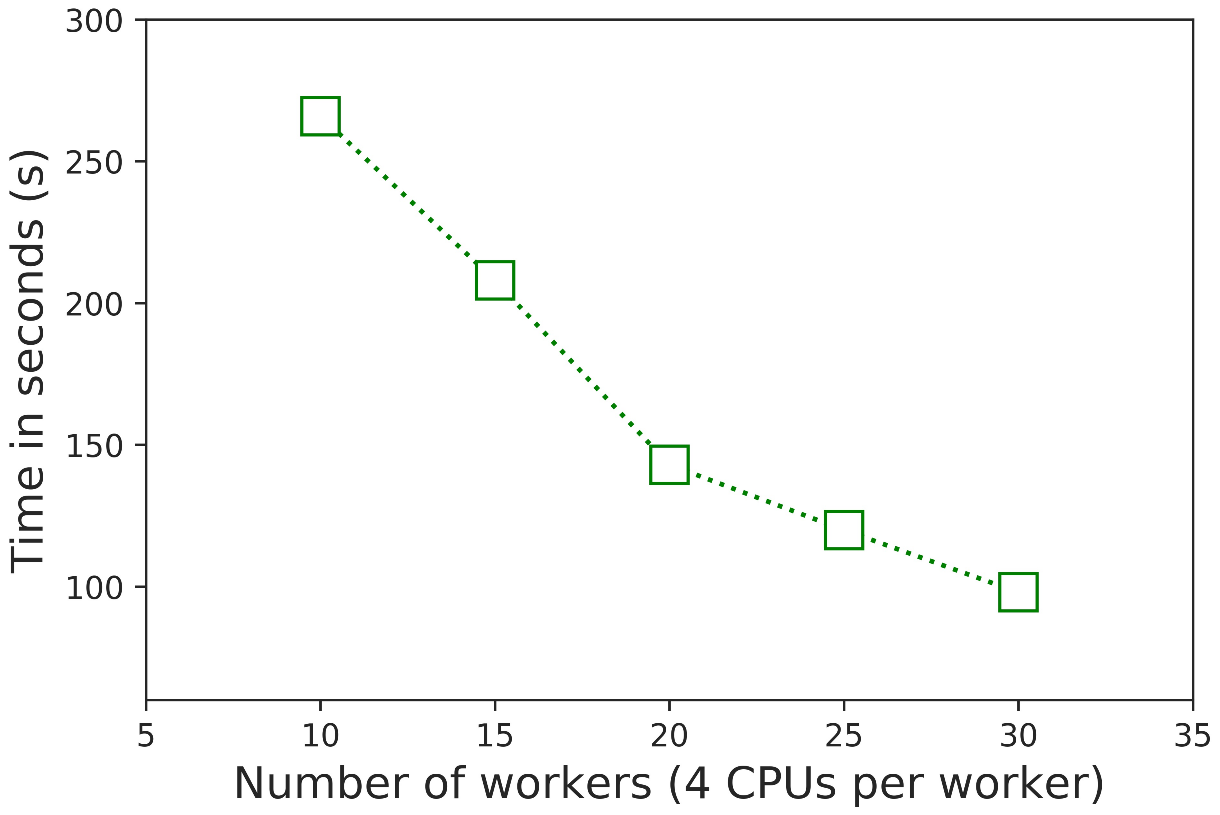
– Newer versions PyIsoP do not return the x, y, z values as 3D grids, this is simply to save space. Although you can use Numpy's meshgrid function to generate these from a grid object 't1' with one line of code

```
import numpy as np
x3d, ,y3d ,z3d = np.meshgrid(t1.x_grid, t1.y_grid, t1.z_grid, indexing='ij')
```

—The grid is calculated over one unit cell. If you'd like to replicate it to say 2x2x3 please use the tile function from Numpy to repeat the grid in blocks.

```
import numpy as np
repeat_grid = np.tile(t1.pot, (2,2,3)) # Let's say you need 2x2x3 for making␣
↪into VTK.
```

—There is this one popular_strategy of using Numba inside Dask to get a massive speed improvement for some algorithms. Unfortunately, the current algorithm requires an array shape change, which prohibits the use of Numba on top of the parallel Dask algorithm. However, the current code is still quite fast. See the benchmarking graph above.



2.3.2 Pore Structure Visualization

The binary vtk file can be used to visualize and elucidate complex pore structures. There are many softwares which can create volume and isosurface rendering from a vtk file. The image below is generated using Visit visualizer. We illustrate the complex pores of ZIF-4 using two isosurfaces at 20000 K (silver) and 0 K (brown).

2.3.3 Energy Histogram

PyIsoP contains the histogram which offers 3 ways for the user to obtain the energy histogram. The number of bins and E_{max} can be set while initializing the histogram. All the energies should be in the units of [K] to ensure consistency with the RASPA grid output.

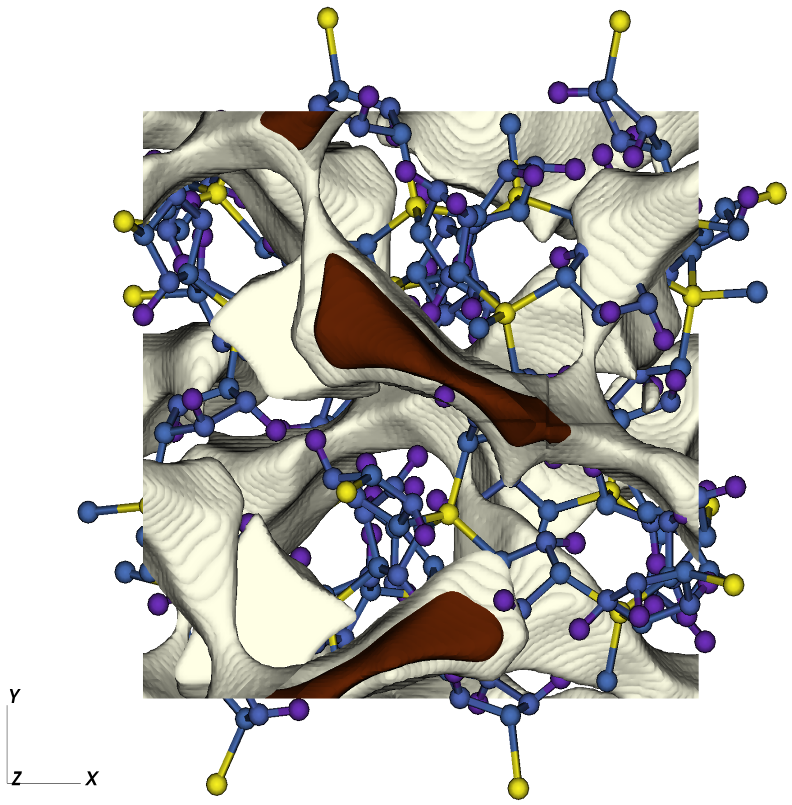
1. From the PyIsoP grid3D object

```
import pyIsoP.histo as histo                    # import the histogram module
h = histo.histo()                                         # initialize a histo object
h = histo.histo.grid2histo(t1, h)               # update (overwrite) the histo object␣
↪with histogram calculated from the grid3D object t2
```

2. Read in the energy grid from a RASPA style .grid file, with x, y, z, E data or from .cube file.

```
import pyIsoP.histo as histo                    # import the histogram module
h = histo.histo()                                         # initialize a histo object
h = histo.histo.raspa2histo('raspa_grid_filename.grid' , ,h)          # update␣
↪(overwrite) the histo object with histogram calculated from the RASPA grid file.
h = histo.histo.cube2histo('cube_filename.cube',h)            # update (overwrite)␣
↪the histo object with histogram calculated from a .cube file
```

(continues on next page)



(continued from previous page)

```
```

3. Read in the histogram as two column text file with no header. Bin-centers in one column, normalized histogram in the other column.

```
import pyIsoP.histo as histo                    # import the histogram module
h = histo.histo()                                        # initialize a histo object
h = histo.histo..file2histo('text_filename.dat', h)             # update (overwrite)␣
↪the histo object with histogram calculated from the RASPA_ grid file.
```

2.3.4 Coordination Number from Machine Learning

In order to predict the guest-guest energy of hydrogen, we use a machine learning model (GPR) trained on the first-shell coordination number. Please refer to *Theoretical Background* section or our recent work by Gopalan *et al.*, [GBBS19] for more details. PyIsoP provides a pre-trained model at 77 K which can predict the hydrogen coordination numbers as a function of log $_{10}$ (P), void fraction, largest cavity diameter (A), pore limiting diameter (A)]

- To load that model (details are in the SI of the publication [GBBS19]

```
import joblib
gp=joblib.load('gprmodel.joblib')                # Load the trained model
n1 = gp.predict([logP, VF, LCD,PLD])            # Predict at 77 K for a set of ␣
↪feature values for log10(pressure), void fraction, LCD and PLD in angstroms.
```

- To train a new model using your own data (at your temperature of choice) but with the default settings using Gaussian Process Regression, create a comma-separated-values (.csv) with 5 columns of “log(P)”, “Vf”, “lcd”, “pld”, “n1” with no header lines. Let’s call it ‘file_with_data.csv’

```
import pyIsoP.machlearn as machlearn

m1= machlearn.machlearn(restarts=2)                          # Initialize object with␣
↪ 2 optimizer restarts
m1 = machlearn.machlearn.GPR4n1( m1, 'file_with_data.csv', 0.9)    # Train the model␣
↪with 90 % training and 10 % Testing
n1 = m1.predict([logP, VF, LCD,PLD])              # Predict at your temperature for a␣
↪set of  feature values for log10(pressure), void fraction, LCD and PLD in angstroms.
```

- Preferred: To use algorithms other than GPR, users are encouraged to train their own model and be ready to provide n_1 as a vector (array corresponding to different pressures) to be fed into the *isotherm* calculation (example below) using the predictor .

2.3.5 Adsorption Isotherm

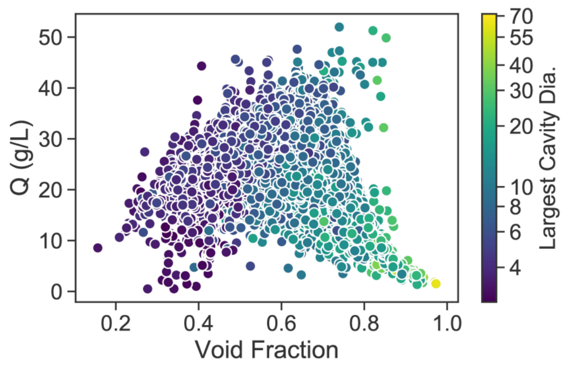
PyIsoP takes in the temperature, pressures, void fraction, the energy histogram object, coordination numbers vector, Lennard-Jones well depth in [K] (should be consistent with the one used in the grid calculation) and the molecular weight (M_A) and predicts the adsorption isotherm in the units of grams per liter of the adsorbent. Combining all the examples from above, the isotherm can be calculated using the predictor as

```
from pyIsoP import predictor
g_L_CH2=predictor.predictors.predict_isotherm(T,P,Vf,h,n1,epsilon=46,MA=14)
```

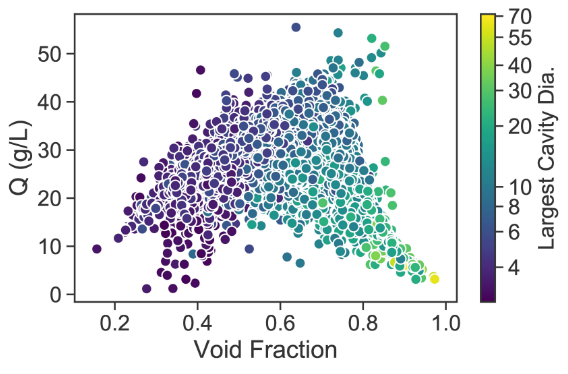
2.3.6 Example Application to High-throughput Screening

CoRE-MOF 2019 All Solvent Removed (12,914 structures)

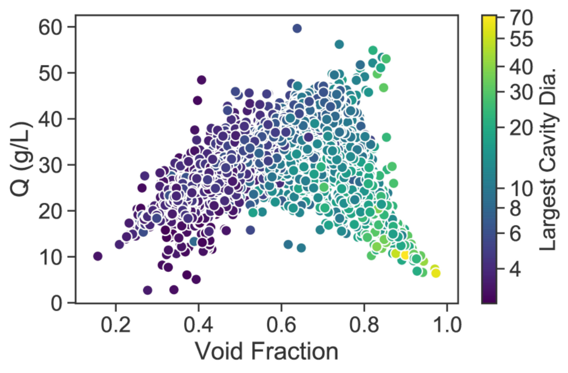
Using the same algorithm implemented as PyIsoP, we calculated the hydrogen adsorption isotherms for a preliminary version of the CoRE MOF 2019-ASR (12,914 structures) from 1 Pa to 100 bar at two temperatures (77 K and 160 K) in less than 24 hrs on 500 processors with a grid spacing of 1 $\AA$. The evolution of the gas uptake for the entire set of 12,914 materials at 77 K with increasing adsorption pressures is depicted in the figures below. Having the entire isotherm enables us to answer important questions regarding maximization of gas uptake quickly and accurately, like determining the choice of the adsorption and desorption conditions for a material with given void fraction and LCD or against any other textural property. For example, consider two materials A (highlighted in yellow in figures (e) and (f)) and B (highlighted in red in figures (e) and (f)) with very similar void fractions, close to 0.85 but with different largest cavity diameters of 13.5 $\AA$ and 34.9 $\AA$, respectively. If one were to use A for storing hydrogen at 77 K, increasing the adsorption pressure from 42 bar ref{fig:L5} to 100 bar ref{fig:L6}) would give an improvement of less than 1% (56.939 g/L to 57.92 g/L) in gas uptake, hence is not worthwhile considering the increased costs and risks of storing at higher pressures. Instead, if one were using B, the same change in pressure will improve the the uptake by about 50% (30.87 g/L to 44.065 g/L), which might be more economically feasible. Please refer to Gopalan *et al.* [GBBS19] for more information.



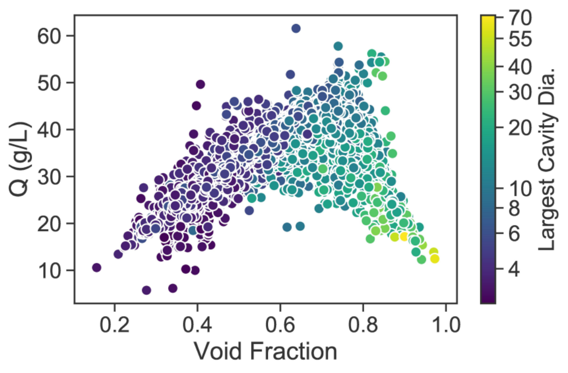
(a) 1.44 bar, 77 K



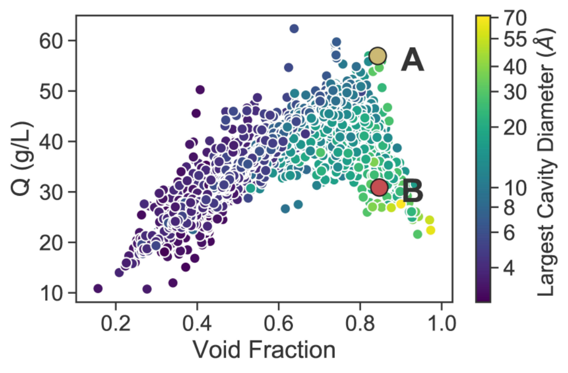
(b) 3.36 bar, 77 K



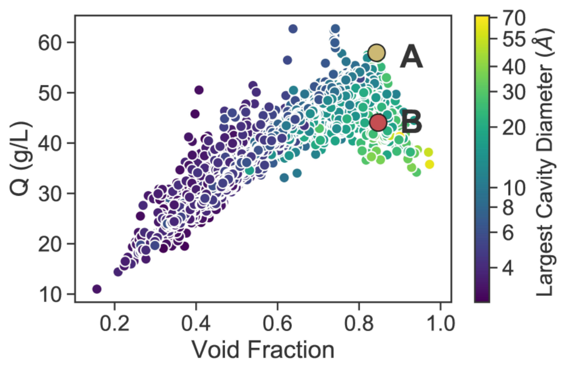
(c) 7.87 bar, 77 K



(d) 18.33 bar, 77 K



(e) 42.81 bar, 77 K



(f) 100 bar, 77 K

2.4 Citing PyIsoP

If you use PyIsoP in your research please consider citing the following article Gopalan *et al.*, [GBBS19]

2.5 Source Code Reference

- genindex
- modindex

** **Acknowledgements:** Andrew Rosen, Snurr Research Group, Northwestern Univerisity. Project based on the Computational Molecular Science Python Cookiecutter version 1.0.

This work is supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences and Biosciences through the Nanoporous Materials Genome Center under award DE-FG02-17ER16362.

Created by: Arun Gopalan

BIBLIOGRAPHY

[CCH+14] Yongchul G Chung, Jeffrey Camp, Maciej Haranczyk, Benjamin J Sikora, Wojciech Bury, Vaiva Krungleviciute, Taner Yildirim, Omar K Farha, David S Sholl, and Randall Q Snurr. Computation-ready, experimental metal–organic frameworks: a tool to enable high-throughput screening of nanoporous crystals. *Chemistry of Materials*, 26(21):6185–6192, 2014.

[GBBS19] Arun G Gopalan, Benjamin J Bucior, Scott N Bobbitt, and Randall Q Snurr. Prediction of hydrogen adsorption in nanoporous materials from the energy distribution of adsorption sites. *Molecular Physics*, ():, 2019. (in press).

[PVG+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. Scikit-learn: machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[Ras04] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[SKL+14] Cory M Simon, Jihan Kim, Li-Chiang Lin, Richard L Martin, Maciej Haranczyk, and Berend Smit. Optimizing nanoporous materials for gas storage. *Physical Chemistry Chemical Physics*, 16(12):5499–5513, 2014.

[SBT94] Randall Q Snurr, Alexis T Bell, and Doros N Theodorou. A hierarchical atomistic/lattice simulation approach for the prediction of adsorption thermodynamics of benzene in silicalite. *The Journal of Physical Chemistry*, 98(19):5111–5119, 1994.

[Was80] Yoshio Waseda. The structure of non-crystalline materials. *Liguids and Amorphous Solids*, 1980.

[GBBS19] Arun G Gopalan, Benjamin J Bucior, Scott N Bobbitt, and Randall Q Snurr. Prediction of hydrogen adsorption in nanoporous materials from the energy distribution of adsorption sites. *Molecular Physics*, ():, 2019. (in press).

[GBBS19] Arun G Gopalan, Benjamin J Bucior, Scott N Bobbitt, and Randall Q Snurr. Prediction of hydrogen adsorption in nanoporous materials from the energy distribution of adsorption sites. *Molecular Physics*, ():, 2019. (in press).