1. Introduction

Optimization techniques have been of significant interest in many different fields: it finds use in fields such as economics and finance, electrical and civil engineering, materials science, and more. Economists use optimization techniques in methods such as control theory to model labor-market behavior, for example, whereas electrical engineers apply optimization to the routing of circuitry. There are many techniques available for finding local minima and maxima of objective functions of one or more variables. In principle, local optimization is relatively straightforward, but not suitable for finding global minimums in systems with many local minima.

Global optimization techniques have been developed to tackle this issue: researchers have implemented techniques such as simulated annealing, Monte Carlo, as well as ones which take advantage of the principles of quantum tunneling. Here we are interested in the **basin hopping** technique, first described by David Wales and Jonathan Doye (**provide reference)**. In this paper, I introduce the fundamental principles behind the basin hopping algorithm in the scipy module and optimize three Leonard-Jones configurations of N=13, 15, and 17 using basin hopping. In doing so, I show the results of adjusting several important parameters in the function and provide an analysis of the parameter values that generated the global minima for the three systems.

1. Theoretical Background

In their work, Wales and Doye tested their basin hopping global optimization method to minimize Leonard-Jones (LJ) potential clusters with many atoms. At the time, techniques such as simulated annealing and those based on quantum tunneling could not sufficiently describe both large (say, N>50) and small (say, N<10) clusters.[[1]](#endnote-1) This motivated Wales and Doye in part, to develop an optimization method that could apply to clusters of N atoms, and whose algorithm could extend to other problems involving potential energy surfaces (PES). LJ potentials were a good test system for developing a technique that is extendable to other systems with complicated PES such as biomolecular systems.[[2]](#endnote-2) A simple Leonard Jones potential is shown below followed by the equation which describes the total potential energy of the configuration.

A picture containing sky

Description automatically generated

**Figure 1:** Leonard Jones potential for a two-atom system showing potential energy as a function of separation distance.

Source: Davoud Raoufi et al. “Study of Carbon Atoms Deposited on Graphene Layer Using Molecular Dynamics Simulation.” AIP Conference Readings, 2019.

The LJ potential is described by the following equation:

where r is the separation distance between each pair of atoms, is the depth of the potential well, and δ represents the distance at which the inter-particle distance is equal to zero. In this paper, both values are taken to be equal to 1. As the number of atoms in the system increase, the number of local minima in the PES drastically increases; thus, selection of appropriate parameter values become increasingly important.

Before moving on to methods and results, I discuss the basin hopping algorithm and how it is implemented in scipy – namely, what parameters I adjust in my experiment, and how these parameters affect the pathway taken toward global optimization. Figure 2 below is a visualization of how the basin-hopping technique works in principle.

A close up of a map

Description automatically generated

**Figure 1**: 2D visualization of the basin-hopping global optimization technique. Staggered “staircases” around each local minimum demonstrate possible locations for “hopping,” where each minimum is optimized and a subsequent step taken until the global minimum is found.

Source: <https://esa.github.io/pagmo2/docs/cpp/algorithms/mbh.html>

In this project, I use scipy’s basin hopping module. This function takes in several parameters, one of them being the function to minimize, and the initial starting positions of (in the case of a LJ potential) the atoms. Below is scipy’s basin hopping function along with its possible parameters one can pass in.

1. **def** basinhopping(func, x0, niter=100, T=1.0, stepsize=0.5,
3. minimizer\_kwargs=None, take\_step=None, accept\_test=None,
5. callback=None, interval=50, disp=False, niter\_success=None,
7. seed=None):

The parameters that I test for global minimum convergence on LJ potentials are the temperature **T**, and the **stepsize**. For the algorithm to function optimally, choosing appropriate values for these two parameters are crucial. I discuss them in more detail below; first, I summarize the basin hopping algorithm. In principle, there are three steps involved in the iterative process.

2.1 The Basin Hopping Algorithm

The basin hopping algorithm is an iterative, stochastic algorithm that has three primary steps:

1. Random perturbation of the current coordinates

We take a random step with an optimal step size applied to each Cartesian direction. The random step in either direction is a value chosen between and . This will place us at or near one of the many local minima within the PES.

1. Local minimization

Next, local minimization is done on the current position. Depending on the system, the user may pass in a minimizer of their choice to the basin hopping algorithm – i.e gradient descent, conjugate gradient, or a quasi-newton method such as L-BFGS. At this step, we “fall” into one of the basins.

1. Accept or reject the new function value.

[1]

The probability of accepting or rejecting the new function value is determined by equation 1, where *T* is the value of the *temperature* parameter. Thus, the acceptance criterion is largely dependent on the temperature value that we choose for the LJ configuration of interest. The value for *T* should be comparable to the difference in *function* value between local minima; I emphasize *function* value because this is not the same as the *height* of the walls between local minima, or basins. This acceptance criterion uses Monte Carlo algorithms and is based on the Metropolis criterion. In determining whether to accept or reject coordinates based on the new function value, first an upper bound is created via the following variable:

[2]

Next, a random number is generated between 0 and 1, using a random number generator.

[3]

If the random number *u* is less than or equal to *w* in equation 2, then the new coordinates and function value are accepted – otherwise, they are rejected. This is the Metropolis criterion.

1. Methods

I write two functions to pass into scipy’s basin hopping function: a position initializer for N-atoms, as well as the total energy function which we wish to minimize. The source code for these functions are shown below.

1. @jit
2. **def** init\_pos(N, L=5):
3. """
4. Intialize starting positions in x,y,z coordinates for N number of atoms.
6. Return: Array with random Cartesian positions of N atoms.
7. """
8. **return** L\*np.random.random\_sample((N\*3,))

11. @jit
12. **def** total\_energy(positions):
13. """
14. Calculate the total energy
16. input:
17. positions: 1\*N array which represents the atomic positions in Cartesian coordinates.
19. output
20. E: the total energy
21. """
22. E = 0
24. N\_atom = int(len(positions)/3)
26. **for** i **in** range(N\_atom-1):
27. **for** j **in** range(i+1, N\_atom):
28. pos1 = positions[i\*3:(i+1)\*3]
29. pos2 = positions[j\*3:(j+1)\*3]
30. dist = np.linalg.norm(pos1-pos2)
31. r6 = dist\*\*6
32. r12 = r6\*r6
33. E += 4\*(1/r12 - 1/r6)
34. **return** E

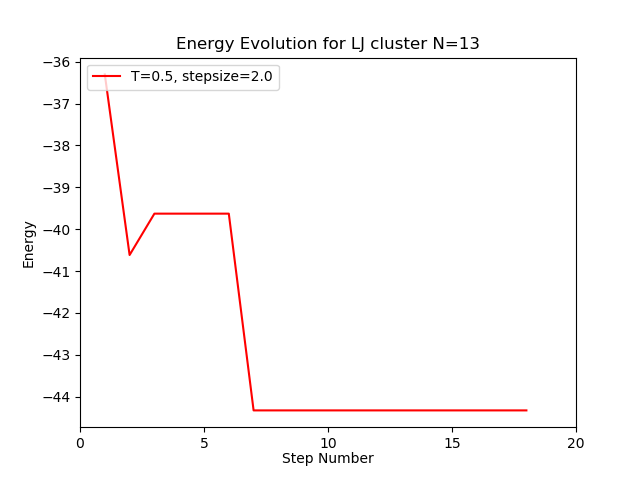
The total\_energy function is the function we wish to minimize using scipy’s basin hopping algorithm. Below, I show the source code for a typical run, using the N=13 atom configuration as an example.

1. true\_13 = -44.326801
2. N\_13 = 13
3. pos\_13 = init\_pos(N\_13)
4. start = time.time()
5. res\_13\_1 = basinhopping(func=total\_energy, x0=pos\_13, niter=20, T=0.5, stepsize=2.0, niter\_success=10, disp=True)
6. **print**("Total time to convergence: ", time.time()-start, 's')
7. **print**("Global minimum value: ", res\_13\_1.fun)
8. **print**("True minimum value: ", true\_13)

In this project, I looked at three LJ configurations: N=13, 15, and 17. First, I tested extreme values for the parameters – a large temperature value with small stepsize, and a large stepsize with small temperature value.

Then, I adjusted the temperature and stepsize parameters until a suitable range was found that successfully found the minimum energy values for each configuration. I kept the number of iterations below 40 for all three configurations. Furthermore, to keep the searches unbiased, I did not input a seed into the basin hopping function for any configuration. Finally, using the atomic positions returned by the basin hopping function, I visualize each LJ structure using a 3D plot.

1. Results & Analysis

A screenshot of a cell phone

Description automatically generatedFirstly, it was obvious that too large and too small of temperature and stepsize values were not sufficient in finding global minima. In either case, the basin hopping function either got “stuck” in a local minimum from which it could not hop out or did not have suitable step size to hop between local minima to the global minimum basin. The results show that there is not one set of temperatures and stepsizes per LJ configuration that lead to global optimization; rather, there is a small range of values that sometimes have success in finding the global minimum. This can be seen in **Table 1**, which shows parameter values that lead to success for each configuration. In Figure 1, I plot the energy evolution as a function of steps for each LJ configuration: here, I show the plots whose parameters had the best success.

Figure 1:Energy Evolution as a function of iterations (steps) for **Left:** N=13 atoms, and **Right:** N=15 atoms. Chosen parameter values show a smooth transition to the global minimum.

1. Wales & Doye… [↑](#endnote-ref-1)
2. Konstantin Order and Davd Wales.. “Mutation Basin-Hopping” [↑](#endnote-ref-2)