1 Introduction

There are many problems in physics which require minimizing some function with respect to some set of variables. Of course, many problems in coursework are designed such that the derivatives of the functions of interest can be analytically manipulated to yield a root, or in the worst case, the root can be obtained easily by utilizing graphical methods. However, practically speaking, most functions are difficult to analytically manipulate in order to yield closed form expressions for the extrema, and when our function depends on a large set of variables, we cannot easily plot the desired function to make direct estimates of the extrema. That is, to some degree, we are blind to the location of a function's extrema when the dimensionality exceeds our three-dimensional intuition.

In particular, one problem of interest is minimizing the energy of a configuration of particles comprising a molecule. The potential energy of a configuration of N particles, which we assume here for the sake of simplicity are of the same type, can be modeled through the Lennard-Jones potential

$$V(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N) = \frac{4\epsilon}{2} \sum_{i \neq j} \left(\frac{\sigma^{12}}{|\boldsymbol{r}_i - \boldsymbol{r}_j|^{12}} - \frac{\sigma^6}{|\boldsymbol{r}_i - \boldsymbol{r}_j|^6} \right). \tag{1}$$

Note that the factor $1/2$ is inserted to counteract overcounting in the sum, ϵ is some parameter that determines the strength of the particle interactions, and σ is a length scale. A plot of the potential between two particles is shown in Fig. 1.

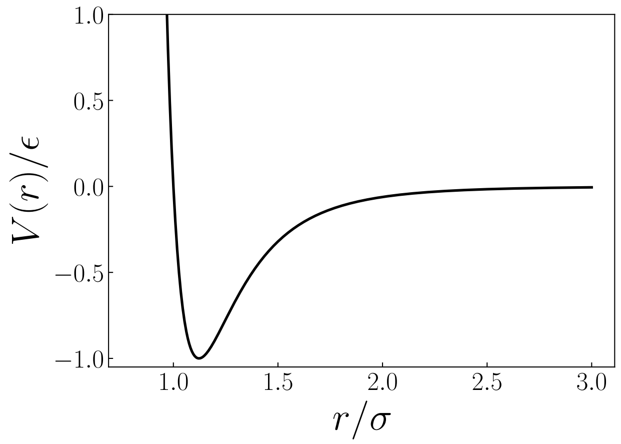

Figure 1: Lennard-Jones potential between two particles as described by Eq. (1).

2 Framework

2.1 Theoretical framework

For the duration of this work, we redefine our potential and length scales as follows:

$$V \to V/\epsilon, \quad r \to r/\sigma. \tag{2}$$

That is, in the rescaled notation, we have

$$V(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N) = 2\sum_{i\neq j}\left(\frac{1}{r^{12}} - \frac{1}{r^6}\right). \tag{3}$$

such that we only deal with dimensionless quantities in our numerical work.

Before describing the numerical methods we use to find the minima of this potential in the case of N-particles, we first explore the analytic solution of this problem. Certainly, one can solve the problem exactly in the case of two particles. The potential

$$V(\boldsymbol{r}_1, \boldsymbol{r}_2) = 4\left(\frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|^{12}} - \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|^6}\right), \tag{4}$$

and if we transform coordinates such that $\boldsymbol{r} = \boldsymbol{r}_1 - \boldsymbol{r}_2$ and $\boldsymbol{R} = (\boldsymbol{r}_1 - \boldsymbol{r}_2)/2$, then

$$V(r) = 4\left(\frac{1}{r^{12}} - \frac{1}{r^6}\right) \Rightarrow V'(r_0) = 4\left(-\frac{12}{r_0^{13}} + \frac{6}{r_0^7}\right) = 0 \Rightarrow r_0 = 2^{1/6}. \tag{5}$$

We remark on a related set of observations which are interesting from a physics standpoint. Observe that the result is a single real number. From the onset, we labeled our potential function as having 4 degrees of freedom labeled by the (x, y) coordinates of each point. Indeed, the potential is translationally invariant, meaning that if we take $\boldsymbol{r}_i \to \boldsymbol{r}_i + \boldsymbol{a}$, the potential does not change, and this stems from the fact that the interaction between particles is described by a central potential. Thus, we have eliminated two degrees of freedom. Additionally, because our potential is central, we can rotate about the z-axis, which is perpendicular to the xy-plane where our particles live, and the angle of rotation eliminates an additional degree of freedom.

One thing that we have not been careful about above is the following. In general, we want to minimize the potential with respect to the positions of each particle, implying the condition $\boldsymbol{\nabla}_i V(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_N)$, where $\boldsymbol{\nabla}_i$ denotes the gradient with respect to $\boldsymbol{r}_i$. Of course, we can translate these conditions in the two particle case under the change of coordinates $\{(x_1, y_1), (x_2, y_2)\}$ to the new coordinates $\{(r, \phi), (R, \Phi)\}$, where $\boldsymbol{r} = r(\cos\phi\hat{\boldsymbol{x}} + \sin\phi\hat{\boldsymbol{y}})$ and $\boldsymbol{R} = R(\cos\Phi\hat{\boldsymbol{x}} + \sin\Phi\hat{\boldsymbol{y}})$ through the chain rule

$$\begin{aligned}
\frac{\partial V}{\partial x_1} &= \frac{\partial r}{\partial x_1}\frac{\partial V}{\partial r} + \frac{\partial R}{\partial x_1}\frac{\partial V}{\partial R} + \frac{\partial \phi}{\partial x_1}\frac{\partial V}{\partial \phi} + \frac{\partial \Phi}{\partial x_1}\frac{\partial V}{\partial \Phi} = 0 \\
\frac{\partial V}{\partial y_1} &= \frac{\partial r}{\partial y_1}\frac{\partial V}{\partial r} + \frac{\partial R}{\partial y_1}\frac{\partial V}{\partial R} + \frac{\partial \phi}{\partial y_1}\frac{\partial V}{\partial \phi} + \frac{\partial \Phi}{\partial y_1}\frac{\partial V}{\partial \Phi} = 0 \\
\frac{\partial V}{\partial x_2} &= \frac{\partial r}{\partial x_2}\frac{\partial V}{\partial r} + \frac{\partial R}{\partial x_2}\frac{\partial V}{\partial R} + \frac{\partial \phi}{\partial x_2}\frac{\partial V}{\partial \phi} + \frac{\partial \Phi}{\partial x_2}\frac{\partial V}{\partial \Phi} = 0 \\
\frac{\partial V}{\partial y_2} &= \frac{\partial r}{\partial y_2}\frac{\partial V}{\partial r} + \frac{\partial R}{\partial y_2}\frac{\partial V}{\partial R} + \frac{\partial \phi}{\partial y_2}\frac{\partial V}{\partial \phi} + \frac{\partial \Phi}{\partial y_2}\frac{\partial V}{\partial \Phi} = 0.
\end{aligned} \tag{6}$$

Note though that V in the transformed coordinates only depends on r, which gives $\partial V/\partial R = \partial V/\partial \phi = \partial V/\partial \Phi = 0$, and therefore, we have transformed our four conditions into a single condition

$$\frac{\partial V}{\partial r} = 0 \tag{7}$$

as we expected from our symmetry considerations above.

Now, if we move onto the case of three particles, then

$$\begin{aligned} V(\boldsymbol{r}_1, \boldsymbol{r}_2, \boldsymbol{r}_3) = 2\Bigg[&\left(\frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|^{12}} - \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_2|^{6}}\right) + \left(\frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_3|^{12}} - \frac{1}{|\boldsymbol{r}_1 - \boldsymbol{r}_3|^{6}}\right) \\ &+ \left(\frac{1}{|\boldsymbol{r}_2 - \boldsymbol{r}_3|^{12}} - \frac{1}{|\boldsymbol{r}_2 - \boldsymbol{r}_3|^{6}}\right)\Bigg]. \end{aligned} \tag{8}$$

Again, we would like to define variables which allows us to retain only the essential degrees of freedom which are uniquely determined at a local minimum of the potential. One may wish to define relative vectors as in the two-particle case, but these generate a linearly dependent set. Another natural, albeit less natural, choice of these new coordinates which contains the same number of independent variables is

$$\boldsymbol{r}_{12} = \boldsymbol{r}_1 - \boldsymbol{r}_2, \quad \boldsymbol{r}_{13} = \boldsymbol{r}_1 - \boldsymbol{r}_3, \quad \boldsymbol{R} = \frac{\boldsymbol{r}_1 + \boldsymbol{r}_2 + \boldsymbol{r}_3}{3}. \tag{9}$$

In these variables, we have

$$V(\boldsymbol{r}_{12}, \boldsymbol{r}_{13}, \boldsymbol{R}) = 2\left[\left(\frac{1}{r_{12}^{12}} - \frac{1}{r_{12}^{6}}\right) + \left(\frac{1}{r_{13}^{12}} - \frac{1}{r_{13}^{6}}\right) + \left(\frac{1}{|\boldsymbol{r}_{12} - \boldsymbol{r}_{13}|^{12}} - \frac{1}{|\boldsymbol{r}_{12} - \boldsymbol{r}_{13}|^{6}}\right)\right]. \tag{10}$$

Herein, though, lies the primary difficulty of analyzing the many-body problem. That is, it is difficult (perhaps impossible) to find a set of coordinates which allow us to decouple to essential degrees of freedom in the minimization condition. Because of time constraints, I have not carried the algebra or calculus any further, but based on the numerical implementation, it may be possible analytically to show that the solution for the three particle configuration is an equilateral triangle. The procedure introduced here for the three-particle configuration can be generalized to the N-particle configuration by introducing the coordinates

$$\boldsymbol{r}_{12} = \boldsymbol{r}_1 - \boldsymbol{r}_2, \quad \ldots, \quad \boldsymbol{r}_{1N} = \boldsymbol{r}_1 - \boldsymbol{r}_N, \quad \boldsymbol{R} = \frac{\boldsymbol{r}_1 + \ldots + \boldsymbol{r}_N}{N}. \tag{11}$$

2.2 Code setup and description

In this section because of the analytic complexity and difficulty of solving for the minimum potential for a configuration of particles, in this section, we discuss a numerical Monte Carlo approach, implemented via a random walk, for finding minima of the potential. Consider an initial configuration $X_0 = \{\boldsymbol{r}_1^{(0)}, \ldots, \boldsymbol{r}_N^{(0)}\}$ of the particles on an $L \times L$ square domain, where

L is some real number discussed further below. We can then define a random walk (X_n) by updating the positions of each particle by the recursive formula

$$\boldsymbol{r}_i^{(n)} = \boldsymbol{r}_i^{(n-1)} + h(2\boldsymbol{u} - 1), \tag{12}$$

where h is another real-valued hyperparameter and $\boldsymbol{u} \in [0, 1]^2$ is a uniform random variable. Thus, the second term implements the random walk by making a step within an $h \times h$ square centered around the previous position. Note, however, that we only accept the new proposed configuration if $V(X_n) < V(X_{n-1})$. While this procedure can be carried out to any precision, we implement three stopping constraints. First, we hard-code a maximum number of iterations in the recursion such that if $n > n_{\max}$, the most recently accepted configuration is returned. Next, we implement absolute and relative precision tolerances δ_a and δ_r, respectively, such that if

$$|V(X_n) - V(X_{n-1})| < \delta_a \text{ or } \left| \frac{V(X_n) - V(X_{n-1})}{V(X_{n-1})} \right| < \delta_r, \tag{13}$$

the iteration is halted, and $V(X_n)$ is returned as output. A *python* implementation of this numerical scheme is displayed in Appendix A.

2.3 A brief discussion of hyperparameters

Note that in the numerical implementation, we are forced to include some non-physical parameters which have an effect on the efficiency, accuracy, and precision of our results, which will be discussed in more detail below. Here, though, we can describe their assigned values. In the code below, we choose $L = 2^{1/6}N$ and $h = 0.2$. We must be careful that our initial L is not too large such that the potential is mostly flat with respect to a given particle's position, which corresponds effectively to a free particle. On the other hand, if we select too small an L, this is less practically problematic, but our particles will eventually diffuse away from each other to a minimum if we choose h well. Note that if h is chosen too large, the particles will tend to make large jumps in their walk and most new positions will be rejected, but on the other hand, if we choose too small an h, our particles will not be able to explore enough of the space to find a global minimum.

While our choice of L is relatively straightforward and only influences the initial configuration directly, the choice of h can be more subtle. Above, we have only concerned ourselves with finding minima, but our goal is actually more specific: we want to find the lowest energy N-particle configuration in the Lennard-Jones potential. That is, we desire to find a global minimum not just a local one, and while there is only one local minimum for $N = 2$ and perhaps $N = 3$, there are more minima for cases with $N > 4$. Indeed, with the simple random walk implementation described above, for $N > 4$, we often find local minima which are not the global minimum.

There are a few strategies one can implement to attempt to find a global minimum. The one implemented here is to run many independent random walks from different starting configurations. By sampling enough distinct starting positions, we may find more than one local minimum, in which case we select the final configuration resulting in the lowest energy. Of course, we may still have the issue that this minimum is not a global minimum, but it is less likely and biased than minimizing on only a single configuration. Additionally, while not done here, this approach is quite amenable to parallelization, which would increase the efficiency of identifying a global minimum and is only then limited by the available hardware given that a single random walk runs fairly quickly. Another method is aimed at escaping local minima. When the walk converges on a local minimum, we can increase the value of h significantly, resulting in an effectively new starting configuration and repeat the random walk process until a local minimum is found. Such a process can be repeated as many times as desired, keeping the configuration corresponding to the lowest energy.

3 Results

With all the theoretical and computational preliminaries dealt with, we can produce some plots. Note that the code implements two approaches. The first method, coined "method 1", minimizes the potential with respect to each particle separately. That is, we first freeze $\boldsymbol{r}_2, \ldots, \boldsymbol{r}_N$ and vary $\boldsymbol{r}_1$ until a stopping condition is met. Then, we freeze $\boldsymbol{r}_1$ and unfreeze $\boldsymbol{r}_2$, varying the latter to minimize the potential. This process is repeated until we have utilized each particle. The second method, coined "method 2", allows the particles to vary in the random walk simultaneously as described in Section 2.2.

Figs. 3–3 demonstrate the results over a single walk for methods 1 and 2 from the same starting configuration on $N = 2$ and $N = 3$. The programs run quite quickly for each of these setups, and it is observed generally that less iterations are required for method 2 to find a minimum.

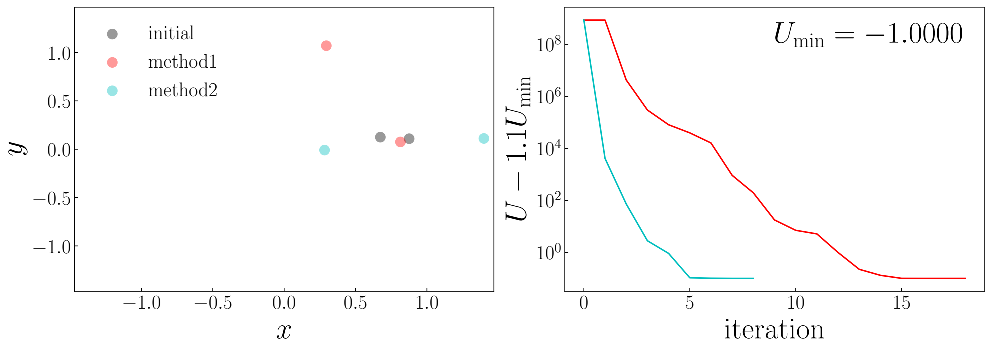

Figure 2: Results of random walk minimization for $N = 2$.

Next, we run the simulation for $N = 4$ as shown in Fig. 4 and Fig. 5. In these cases, we already run into the issue of finding local minima.

In Fig. 6, we display the results of running a batch of minimizations, and as desired, we are able to separate the global minimum from the ensemble of local minima. Note, though, that the current implementation in a for loop is a very crude and inefficient one. To generate Fig. 6, we sampled 100 different initial configurations, and it took nearly 20 minutes on

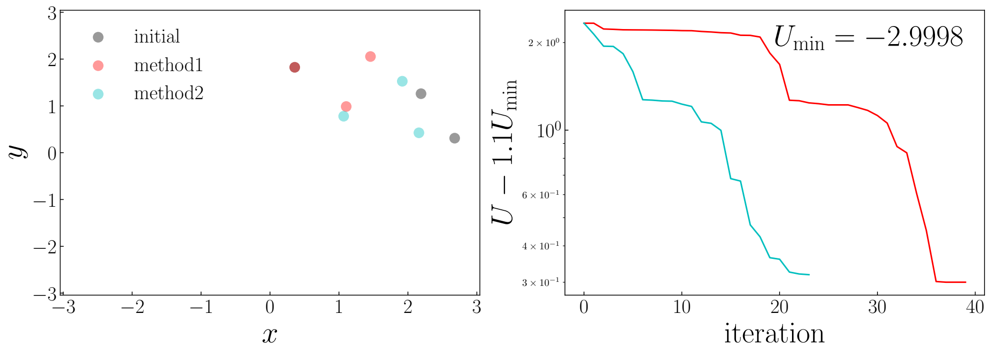

Figure 3: Results of random walk minimization for $N = 3$.

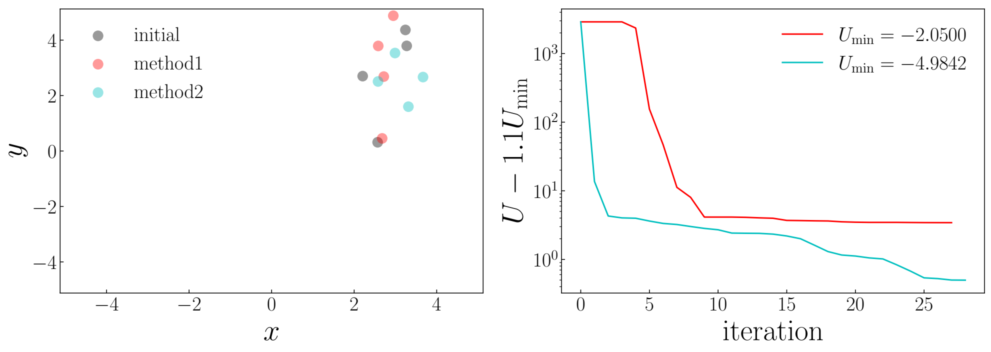

Figure 4: Results of random walk minimization for $N = 4$.

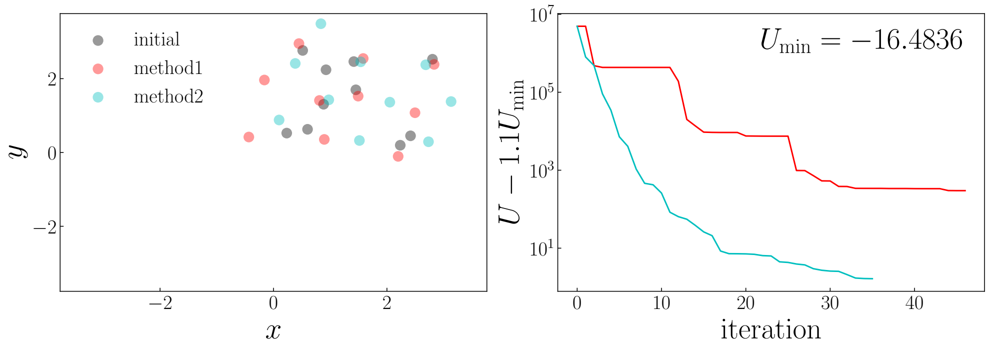

Figure 5: Results of random walk minimization for $N = 10$.

my Mac laptop to minimize all the configurations, although, on average takes less than a minute per configuration to minimize. As discussed above, this approach should ideally be parallelized to increase efficiency.

4 Conclusion

In this article, we have described the minimization of the Lennard-Jones potential for a system of N-particles, analytically and numerically via a random walk Monte Carlo procedure. While the implementation of the numerical scheme is somewhat crude, we have discussed a

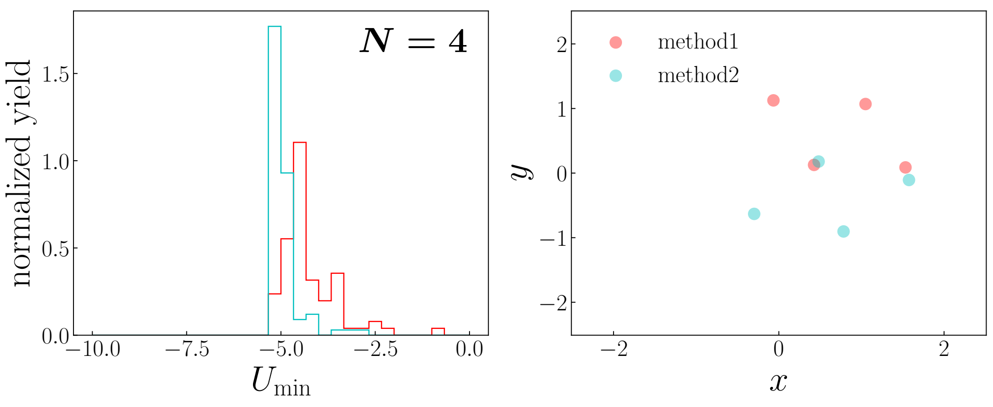

Figure 6: Results of a batch random walk minimization for $N = 4$.

few possible adjustments to increase its efficiency. Alternatively, one could also implement other methods to minimize the potential. Within the class of Monte Carlo methods, we could utilize some importance sampling or Markov chain method, which would sample the potential and allow us to explore the configuration space in a different way. There are also minimization procedures that are not based on Monte Carlo methods such gradient descent. Such methods require more knowledge of the function being minimized or additional numerical schemes to be implemented, but may be better suited for this problem. Whichever scheme is implemented, there are associated unique issues, but the issue of becoming stuck in local minima is a ubiquitous one, requiring careful consideration in a more serious research setting.

A Code Listing

Main python script

```
import numpy as np
from tqdm import tqdm,trange

def get_LJ_potential(P,sig=1,eps=1):
    x = P.T[0]
    y = P.T[1]

    X = np.tile(x,(x.shape[0],1))
    XT = X.T

    Y = np.tile(y,(y.shape[0],1))
    YT = Y.T

    r = np.sqrt((X - XT)**2 + (Y - YT)**2).flatten()
    r = r[r!=0]

    return np.sum(4*eps*((sig/r)**12 - (sig/r)**6))/2


def walk(config,h=1,freeze=None):
    step = h*(2*np.random.uniform(size=config.shape)-1)

    if freeze is not None:
        cond = np.tile(freeze,(2,1)).T
        step *= cond

    return config + step

def minimize(config,h=1,sig=1,eps=1,rtol=1e-3,atol=1e-10,max_iter=int(1e8),
    freeze=None,track=False):
    configs = [config]
    U = [get_LJ_potential(config)]

    for i in range(max_iter):
        new_config = walk(configs[-1],h,freeze)
        new_U = get_LJ_potential(new_config,sig,eps)

        if new_U > U[-1]:
            continue

        configs.append(new_config)
        U.append(new_U)

        adiff = np.abs(new_U - U[-2])
        rdiff = np.abs(adiff/U[-2])
        if adiff < atol or rdiff < rtol:
            break

    if track:
        return np.array(configs),np.array(U)
    else:
        return configs[-1],U[-1]

def minimize1(config,h=1,sig=1,eps=1,rtol=1e-3,atol=1e-10,max_iter=int(1e8),
    track=False):
    configs,U = [config],[get_LJ_potential(config)]
    N = config.shape[0]
    for i in range(N):
        freeze=np.array([j==i for j in range(N)])
        args = (h,sig,eps,rtol,atol,max_iter,freeze,True)
        configs_,U_ = minimize(configs[-1],*args)
        configs += list(configs_)
        U       += list(U_)

    if track:
        return np.array(configs),np.array(U)
    else:
        return configs[-1],U[-1]
```

Script for single configuration minimization

```
import numpy as np
from tqdm import tqdm,trange
```

```
import matplotlib.pyplot as plt
plt.rcParams.update({
    'text.latex.preamble': r'\usepackage{amsmath}',
    'text.usetex': True,
    'font.family': 'sans-serif',
    'font.sans-serif': ['Helvetica']
})

from main import *

h = 0.2
N = 4
L = N*2**(1/6)
P = L*np.random.uniform(size=(N,2))

results = {'method 1': {}, 'method 2': {}}
configs,U = minimize1(P,h,track=True)
results['method 1']['configs'] = configs
results['method 1']['U'] = U

configs,U = minimize(P,h,track=True)
results['method 2']['configs'] = configs
results['method 2']['U'] = U


nrows,ncols=1,2
fig,ax=plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,5*nrows))

x,y = configs[0].T
ax[0].scatter(x,y,marker='o',s=75,lw=2,color='k',alpha=0.4,edgecolor='k',label
    =r'$\rm initial$')

colors = {'method 1': 'r', 'method 2': 'c'}
Umin = min(results['method 1']['U'][-1],results['method 2']['U'][-1])
for _ in results:
    x,y = results[_]['configs'][-1].T
    ax[0].scatter(x,y,marker='o',s=75,lw=2,color=colors[_],alpha=0.4,edgecolor
    =colors[_],label=r'$\rm %s$'%_)

    U = results[_]['U']
    ax[1].plot(U-1.1*Umin,color=colors[_],label=r'$U_{\rm min} = %.4f$'%U[-1])

ax[0].legend(fontsize=20,loc='upper left',frameon=False)
ax[0].set_xlabel(r'$x$',size=30)
ax[0].set_ylabel(r'$y$',size=30)

scale = 1.05*max(np.amax(np.abs(results['method 1']['configs'])),np.amax(np.
    abs(results['method 2']['configs'])))
ax[0].set_xlim(-scale,scale)
ax[0].set_ylim(-scale,scale)


ax[1].set_xlabel(r'$\rm iteration$',size=30)
```

```
ax[1].set_ylabel(r'$U - 1.1 U_{\rm min}$',size=30)
ax[1].semilogy()
ax[1].legend(frameon=False,fontsize=20)

for i in range(2):
    ax[i].tick_params(axis='both',which='major',direction='in',labelsize=20)
ax[1].tick_params(axis='both',which='major',direction='in',labelsize=20)

plt.tight_layout()
plt.show()
fig.savefig(r'track_min%d.pdf'%N,bbox_inches='tight')
```

Script for batch configuration minimization

```
import numpy as np
from tqdm import tqdm,trange

import matplotlib.pyplot as plt
plt.rcParams.update({
    'text.latex.preamble': r'\usepackage{amsmath}',
    'text.usetex': True,
    'font.family': 'sans-serif',
    'font.sans-serif': ['Helvetica']
})

from main import *

h = 0.2
N = 4
L = N*2*(1/6)

results = {'method 1': {'configs': [], 'U': []}, 'method 2': {'configs': [], '
   U': []}}
for i in trange(100):
    P = L*np.random.uniform(size=(N,2))
    configs,U = minimize1(P,h,max_iter=int(1e3),track=False)
    results['method 1']['configs'].append(configs)
    results['method 1']['U'].append(U)

    configs,U = minimize(P,h,max_iter=int(1e6),track=False)
    results['method 2']['configs'].append(configs)
    results['method 2']['U'].append(U)

for _1 in results:
    for _2 in results[_1]:
        results[_1][_2] = np.array(results[_1][_2])


nrows,ncols=1,2
fig,ax = plt.subplots(nrows=nrows,ncols=ncols,figsize=(7*ncols,5*nrows))

colors = {'method 1': 'r', 'method 2': 'c'}
for _ in results:
    idxs = np.argsort(results[_]['U'])
    configs = results[_]['configs'][idxs]
    U = results[_]['U'][idxs]
    counts,edges = np.histogram(U,bins=30,range=(-10,0),density=True)
    ax[0].stairs(counts,edges,color=colors[_])

    x,y = configs[0].T
    ax[1].scatter(x,y,marker='o',s=75,lw=2,color=colors[_],alpha=0.4,edgecolor
   =colors[_],label=r'$\rm %s$'%_)

ax[0].set_xlabel(r'$U_{\rm min}$',size=30)
ax[0].set_ylabel(r'$\rm normalized~yield$',size=30)
ax[0].text(s=r'\boldmath $N = %d$'%N,size=30,x=0.95,y=0.95,va='top',ha='right'
    ,transform=ax[0].transAxes)

ax[1].set_xlabel(r'$x$',size=30)
ax[1].set_ylabel(r'$y$',size=30)

scale = 1.05*max(np.amax(np.abs(results['method 1']['configs'])),np.amax(np.
    abs(results['method 2']['configs'])))
ax[1].set_xlim(-scale,scale)
ax[1].set_ylim(-scale,scale)
ax[1].legend(frameon=False,fontsize=20,loc='upper left')

for i in range(2):
    ax[i].tick_params(axis='both',which='major',labelsize=20,direction='in')

plt.show()
fig.savefig(r'batch_min%d.pdf'%N,bbox_inches='tight')
```