

hard-disk-MC

October 12, 2026

1 2D hard disk Monte Carlo

MC code, similar to Metropolis *et al.* to simulate hard disks in 2D.

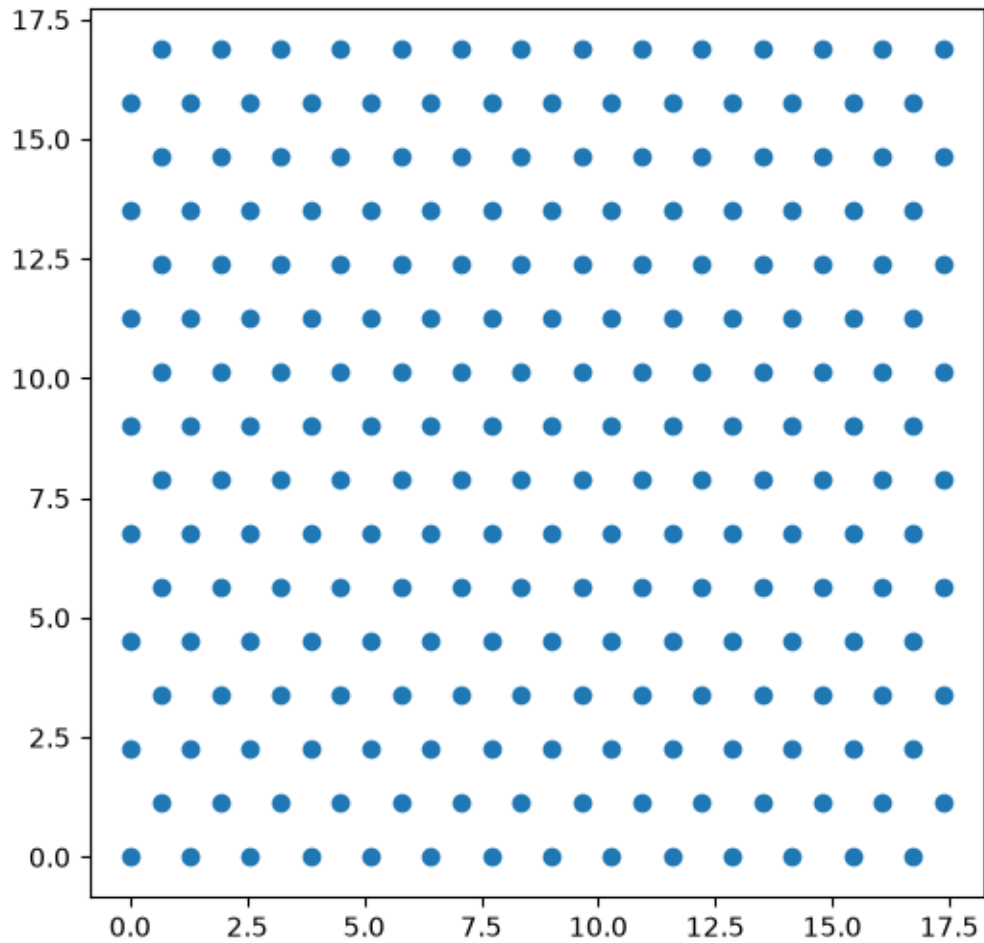
```
[1]: import numpy as np
      rng = np.random.default_rng()
      import matplotlib.pyplot as plt
      plt.rcParams['figure.figsize'] = [6, 6]
      from numba import jit
      from tqdm.auto import trange
```

```
[2]: rc = 0.5
      L, Nx, Ny = 18, 14, 16
      N = Nx*Ny
      V = L*L/(N*np.pi*rc**2)
      rho = 1.0/V # fill fraction
      rho, V, 0.5*np.sqrt(V)
```

```
[2]: (0.5429913228426803, 1.841650055777646, np.float64(0.6785370394786209))
```

```
[3]: # initialize our particles on a square lattice, staggered
      r_i = np.zeros((N, 2))
      n = 0
      for i in range(Nx):
          for j in range(Ny):
              r_i[n] = (i+0.5*(j%2))*(L/Nx), j*(L/Ny)
              n += 1
```

```
[4]: plt.plot(r_i[:,0], r_i[:,1], 'o');
```



```
[5]: @jit(nopython=True)
def MC(pos, rc, i_rand, delta, L):
    """Run MC for hard disks where we attempt to move atom i_rand by delta.
    Returns acceptance ratio, and overwrites pos in place. PBC with
    sidelength L; uses N^2 method for checking distance"""
    N = len(pos)
    accept = 0
    rc2 = 4*rc*rc
    L2 = L*0.5
    for n in range(len(i_rand)):
        i = i_rand[n]
        rp = pos[i] + delta[n]
        collide = False
        for j in range(N):
            if j == i: continue
            r_ij = pos[j] - rp
            if (r_ij[0] < -L2): r_ij[0] += L
```

```

        if (r_ij[0] > L2): r_ij[0] -= L
        if (r_ij[1] < -L2): r_ij[1] += L
        if (r_ij[1] > L2): r_ij[1] -= L
        if (r_ij[0]*r_ij[0] + r_ij[1]*r_ij[1]) < rc2:
            collide = True
            break
    if not collide:
        accept += 1
        if (rp[0] < 0): rp[0] += L
        if (rp[0] > L): rp[0] -= L
        if (rp[1] < 0): rp[1] += L
        if (rp[1] > L): rp[1] -= L
        pos[i,0] = rp[0]
        pos[i,1] = rp[1]
    return accept/len(i_rand)

```

```

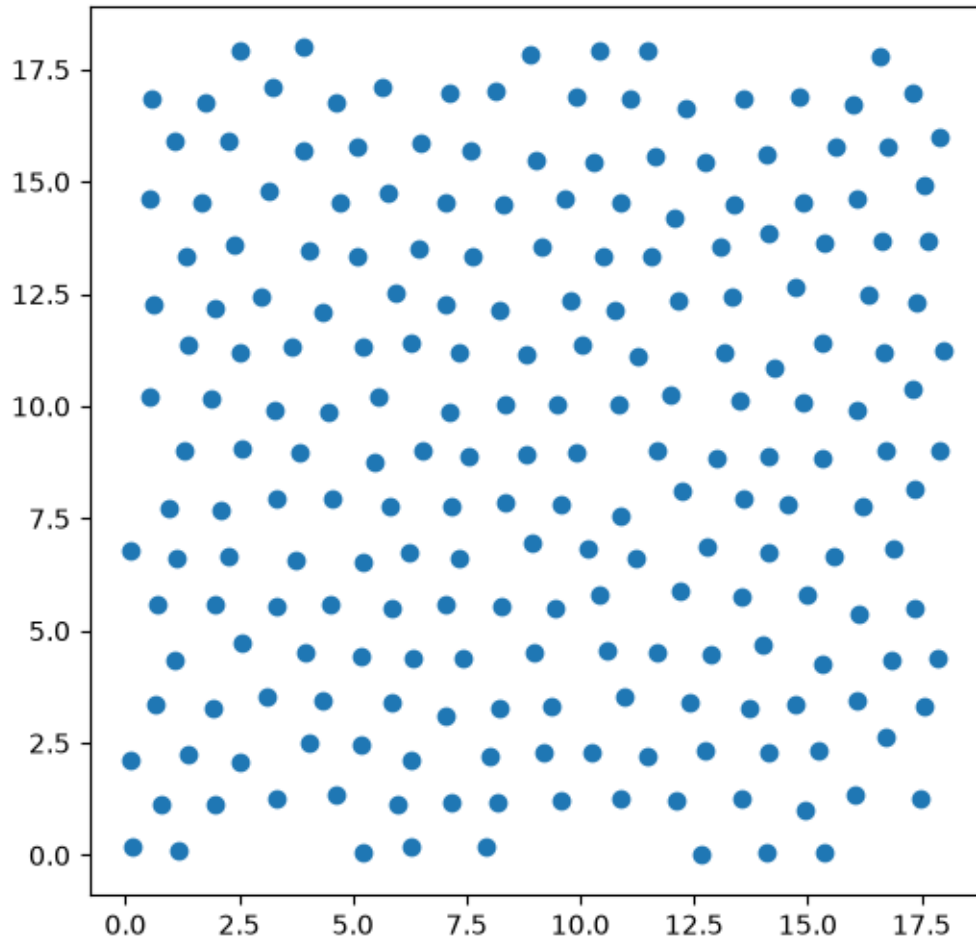
[6]: N_MC = 1024
Delta = 0.5*np.sqrt(V)-rc
i_rand = rng.choice(N, size=N_MC)
delta = rng.uniform(-Delta, Delta, size=(N_MC, 2))
accept = MC(r_i, rc, i_rand, delta, L)

```

```

[7]: plt.plot(r_i[:,0], r_i[:,1], 'o');

```



```
[8]: accept
```

```
[8]: 0.7373046875
```

```
[9]: @jit(nopython=True)
def g_r(pos, L, g_bins, r2_cut):
    """Take in positions, add to histograms"""
    N = len(pos)
    Nr = len(r2_cut)
    L2 = L/2
    for i in range(N):
        for j in range(i):
            r_ij = pos[i] - pos[j]
            if (r_ij[0] < -L2): r_ij[0] += L
            if (r_ij[0] > L2): r_ij[0] -= L
            if (r_ij[1] < -L2): r_ij[1] += L
            if (r_ij[1] > L2): r_ij[1] -= L
```

```

    r2 = (r_ij[0]*r_ij[0] + r_ij[1]*r_ij[1])
    for n in range(1,Nr):
        if r2_cut[n-1] < r2 < r2_cut[n]:
            g_bins[n] += 1

```

```

[10]: rhist = np.linspace(0,L/2,num=100)
      r2_cut = rhist*rhist
      g_bins = np.zeros_like(rhist)

```

```

[11]: g_r(r_i, L, g_bins, r2_cut)

```

```

[12]: g_bins

```

```

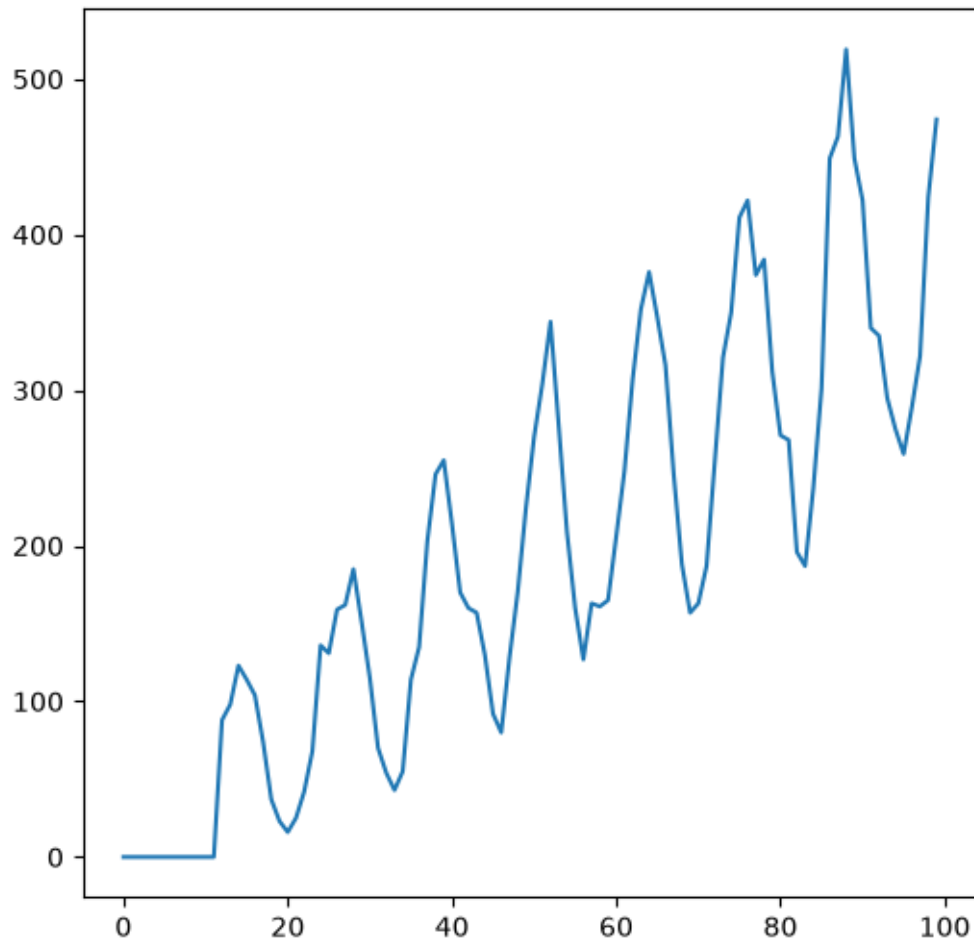
[12]: array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
            0., 88., 98., 123., 114., 104., 74., 37., 23., 16., 25.,
            42., 68., 136., 131., 159., 162., 185., 151., 115., 70., 54.,
            43., 55., 114., 135., 203., 246., 255., 214., 170., 160., 157.,
            130., 92., 80., 128., 170., 224., 270., 304., 344., 277., 209.,
            160., 127., 163., 161., 165., 206., 248., 307., 352., 376., 347.,
            316., 246., 188., 157., 163., 187., 254., 321., 350., 411., 422.,
            374., 384., 312., 271., 268., 196., 187., 237., 301., 449., 463.,
            519., 449., 422., 340., 335., 295., 275., 259., 289., 322., 424.,
            474.])

```

```

[13]: plt.plot(g_bins);

```



```
[14]: N_sweeps = 1024
N_MC = 1024
Delta = 0.5*np.sqrt(V)-rc

rhist = np.linspace(0,L/2,num=100)
r2_cut = rhist*rhist
g_bins = np.zeros_like(rhist)

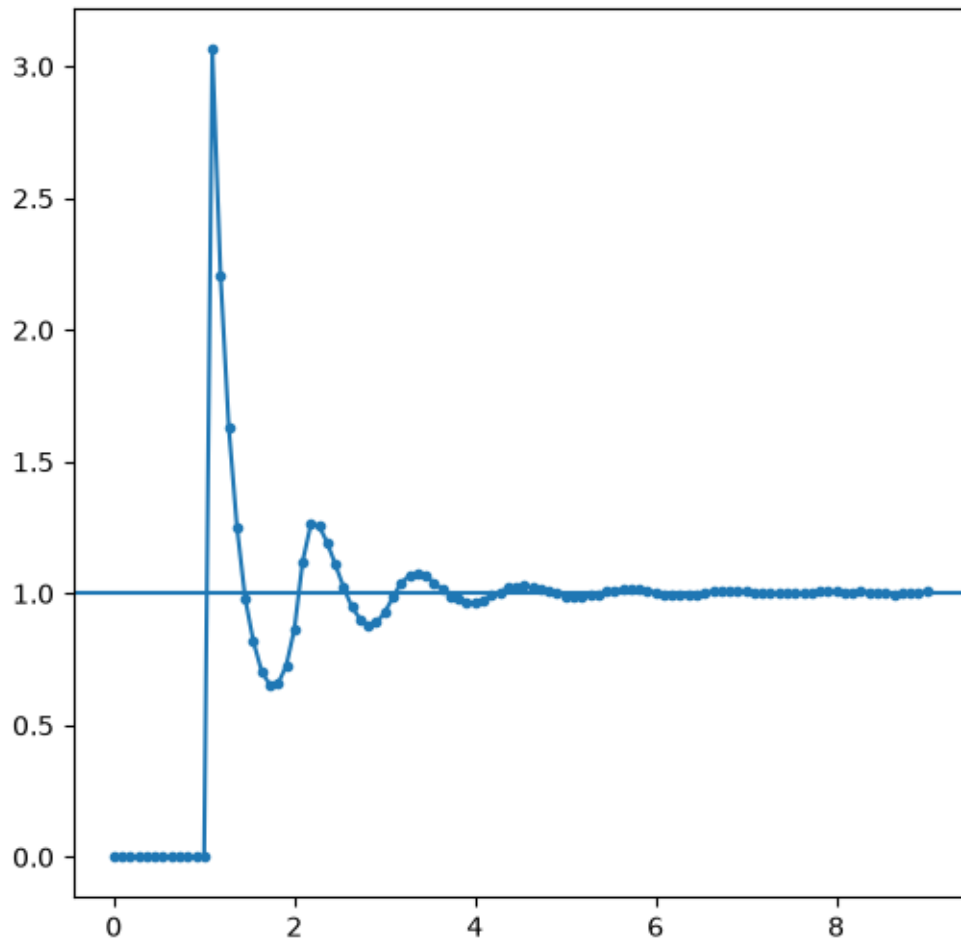
accept_ave = 0
for _ in range(N_sweeps):
    i_rand = rng.choice(N, size=N_MC)
    delta = rng.uniform(-Delta, Delta, size=(N_MC, 2))
    accept_ave += MC(r_i, rc, i_rand, delta, L)
    g_r(r_i, L, g_bins, r2_cut)
g_bins /= N_sweeps
accept_ave /= N_sweeps
accept_ave
```

0%| | 0/1024 [00:00<?, ?it/s]

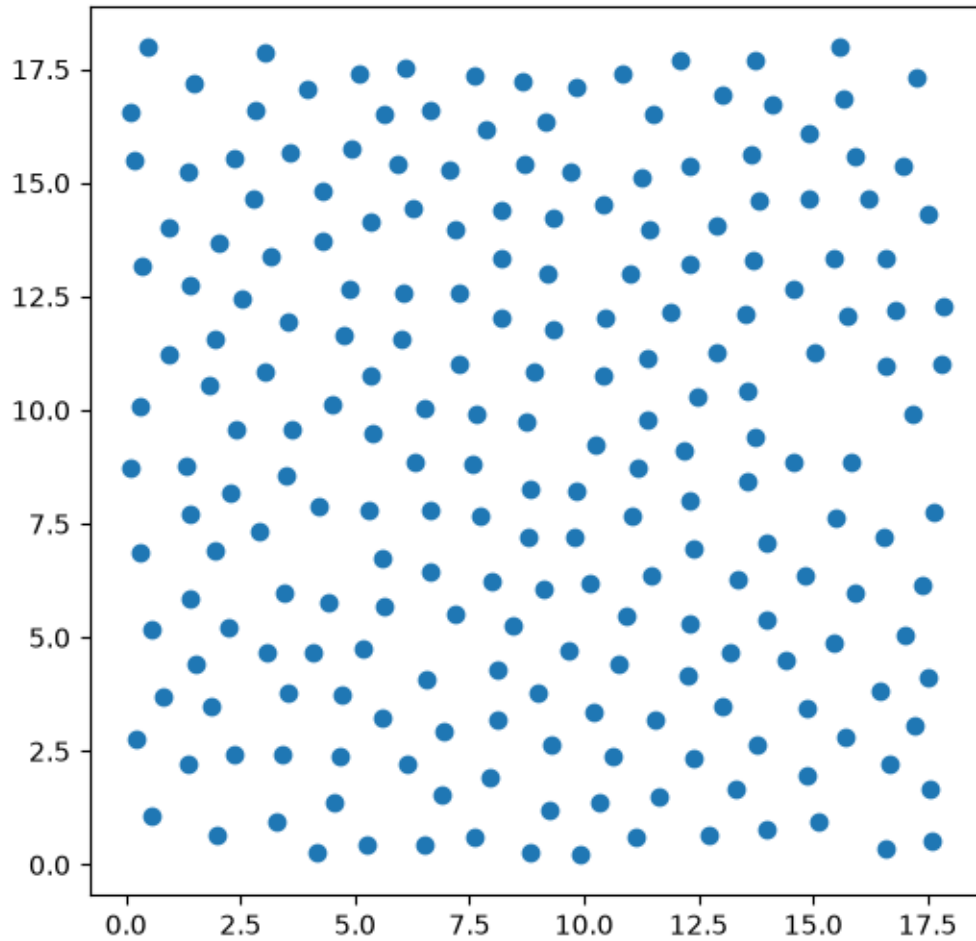
[14]: 0.5227375030517578

```
[15]: # normalize g_bins:
      for n in range(len(g_bins)-1):
          g_bins[n+1] /= 0.5*N*(N-1)*np.pi*(r2_cut[n+1]-r2_cut[n])/(L*L)
```

```
[16]: plt.plot(rhist, g_bins, '-');
      plt.axhline(1.0);
```



```
[17]: plt.plot(r_i[:,0], r_i[:,1], 'o');
```



Now, let's scan through some different different supercell sizes to see how $g(r)$ changes.

```
[18]: g_r_dict = {18.0: g_bins.copy()}
L_range = (17.5, 17.0, 16.5, 16.0, 15.5, 15.0, 14.5)
L_old = L
for L in L_range:
    # scale positions down to fit in box
    r_i *= L/L_old
    Delta *= L/L_old # do smaller steps too
    g_bins = np.zeros_like(rhist)

    accept_ave = 0
    for _ in trange(N_sweeps):
        i_rand = rng.choice(N, size=N_MC)
        delta = rng.uniform(-Delta, Delta, size=(N_MC, 2))
        accept_ave += MC(r_i, rc, i_rand, delta, L)
        g_r(r_i, L, g_bins, r2_cut)
```

```

g_bins /= N_sweeps
accept_ave /= N_sweeps
# normalize g_bins:
for n in range(len(g_bins)-1):
    g_bins[n+1] /= 0.5*N*(N-1)*np.pi*(r2_cut[n+1]-r2_cut[n])/(L*L)
g_r_dict[L] = g_bins.copy()
L_old = L
print(L, accept_ave)

```

```

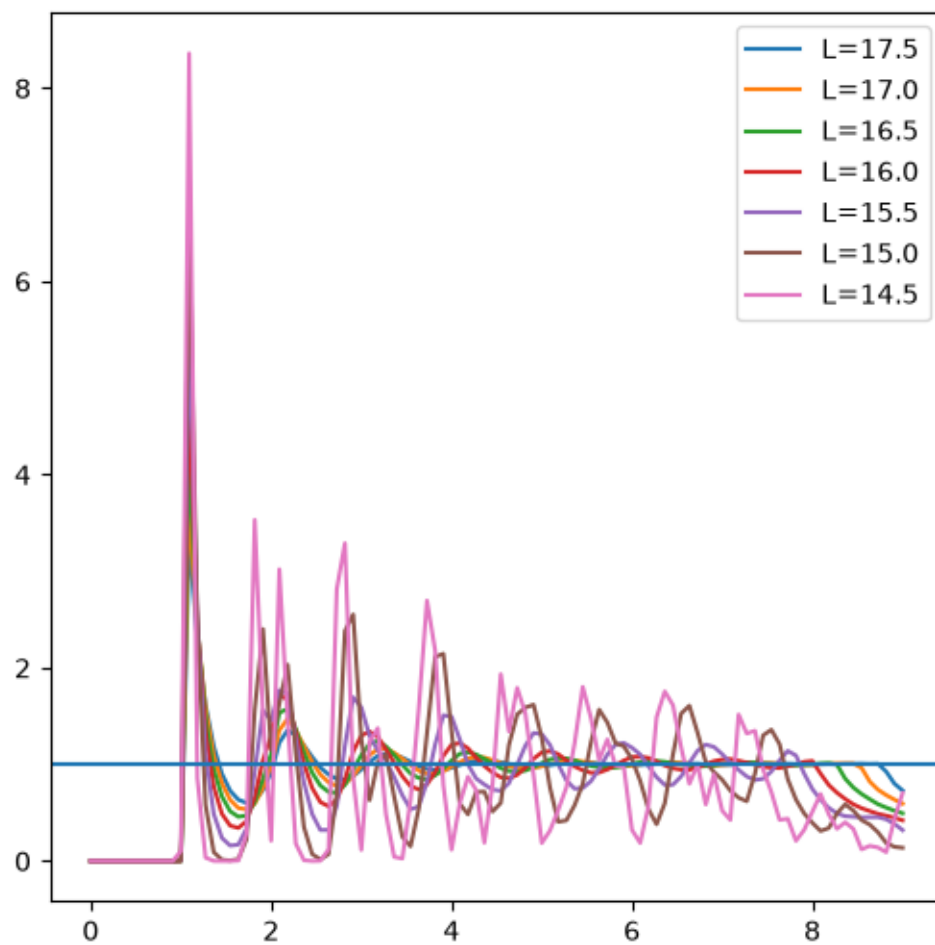
0%|          | 0/1024 [00:00<?, ?it/s]
17.5 0.4718799591064453
0%|          | 0/1024 [00:00<?, ?it/s]
17.0 0.4166412353515625
0%|          | 0/1024 [00:00<?, ?it/s]
16.5 0.345855712890625
0%|          | 0/1024 [00:00<?, ?it/s]
16.0 0.27237987518310547
0%|          | 0/1024 [00:00<?, ?it/s]
15.5 0.1988840103149414
0%|          | 0/1024 [00:00<?, ?it/s]
15.0 0.12871646881103516
0%|          | 0/1024 [00:00<?, ?it/s]
14.5 0.03182029724121094

```

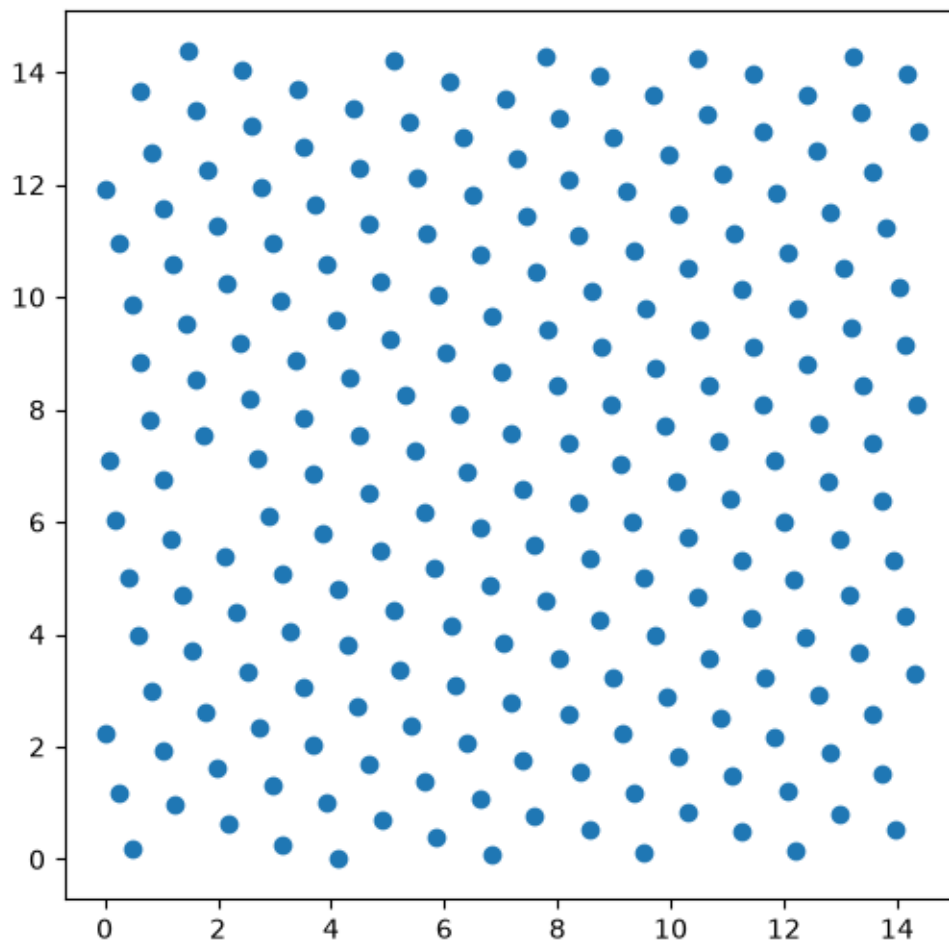
```

[19]: for L in L_range[:]:
    plt.plot(rhist, g_r_dict[L], label=f"L={L}");
plt.legend();
plt.axhline(1.0);

```



```
[20]: plt.plot(r_i[:,0], r_i[:,1], 'o');
```



[]: