

0.1 Thermal calculations on images

```
1 import numpy as np
2 from scipy import interpolate, special

1 import sys
2 if 'src' not in sys.path: sys.path.append('src')
3 import wanglandau as wl
```

0.1.1 Parallel Simulation

```
1 N = 16
2 M = 2**5 - 1
3 I0 = np.zeros(N, dtype=int)
4 system_params = {
5     'StatisticalImage': {
6         'I0': I0,
7         'I': I0.copy(),
8         'M': M
9     }
10 }

1 # L = 16
2 # system_params = {
3 #     'IsingModel': {
4 #         'spins': np.ones((L, L), dtype=int)
5 #     }
6 # }

1 params = {
2     'system': system_params,
3     'simulation': {
4         'max_sweeps': 500_000_000,
5         'flat_sweeps': 10_000,
6         'eps': 1e-8,
7         'logf0': 1,
8         'flatness': 0.1
9     },
10    'parallel': {
11        'bins': 8,
12        'overlap': 0.25,
13        'sweeps': 1_000_000
14    },
15    'save': {
16        'prefix': 'simulation-',
17        'dir': 'data'
18    }
19 }

1 params.pop('parallel', None) # Single run
2 wresults = wl.run(params, log=True)
```

Run parameters

```
-----
{'system': {'StatisticalImage': {'I0': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]),
                                'I': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])},
```

```

        'M': 31}},
'simulation': {'max_sweeps': 500000000,
               'flat_sweeps': 10000,
               'eps': 1e-08,
               'logf0': 1,
               'flatness': 0.1},
'save': {'prefix': 'simulation-', 'dir': 'data'},
'log': True}

```

Running ...

Wang-Landau START

fiter	steps	max steps
-----	-----	-----
1	1120000	4852245278
2	480000	6230406265
3	800000	7059975221
4	640000	7515304503
5	960000	7753865876
6	1600000	7875971497
7	1760000	7937743507
8	2240000	7968810956
9	2080000	7984390249
10	3520000	7992191314
11	4800000	7996094704
12	9600000	7998047114
13	10880000	7999023498
14	17760000	7999511734
15	27200000	7999755864
16	35040000	7999877931
17	21280000	7999938966
18	67200000	7999969483
19	32480000	7999984742
20	142400000	7999992371
21	104800000	7999996186
22	24000000	7999998093
23	288480000	7999999047
24	29280000	7999999524
25	38560000	7999999762
26	41600000	7999999881
27	97120000	7999999941
28	28960000	7999999971

Done: 1036640000 total MC iterations; converged.

... done in 116 seconds.

Writing results ... done: data/simulation-gozi5xqv.h5

```

1  [r['converged'] for r in wldata['results']]

```

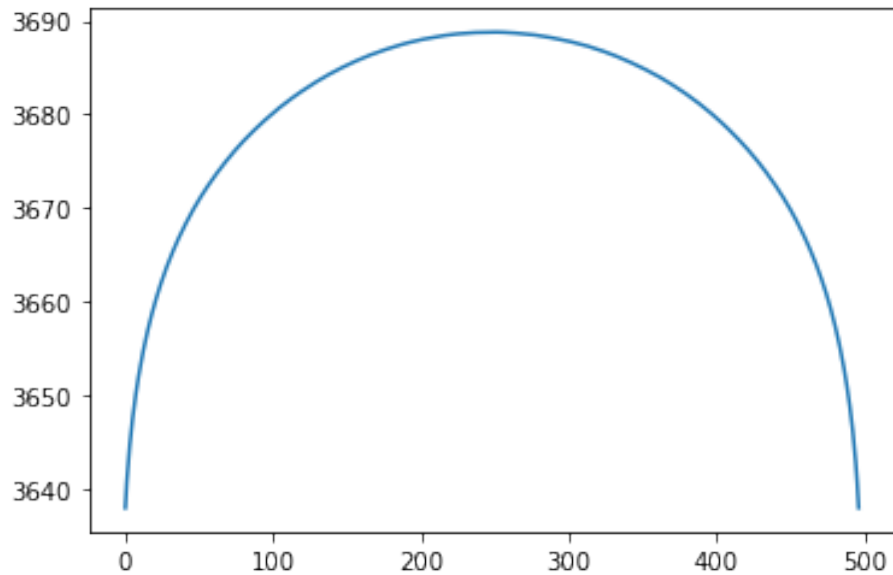
```
[True]
```

```
1 wEs, S, ΔS = wl.join_results(wlresults['results'])
```

0.1.2 Results

```
1 import matplotlib.pyplot as plt
```

```
1 for i, r in enumerate(wlresults['results']):  
2     plt.plot(r['Es'][:-1], r['S'] + ΔS[i])
```



```
1 N, M = len(system_params['StatisticalImage']['I0']), system_params['StatisticalImage']['M']
```

Fit a spline to interpolate and optionally clean up noise, giving WL g 's up to a normalization constant.

```
1 gspl = interpolate.splrep(wEs, S, s=0*np.sqrt(2))  
2 wlgs = np.exp(interpolate.splev(wEs, gspl) - min(S))
```

0.1.3 Exact density of states

We only compute to halfway since g is symmetric and the other half's large numbers cause numerical instability.

```
1 def reflect(a, center=True):  
2     if center:  
3         return np.hstack([a[:-1], a[-1], a[-2::-1]])  
4     else:  
5         return np.hstack([a, a[::-1]])
```

The exact density of states for uniform values. This covers the all gray and all black/white cases. Everything else (normal images) are somewhere between. The gray is a slight approximation: the ground level is not degenerate, but we say it has degeneracy 2 like all the other sites. For the numbers of sites and values we are using, this is insignificant.

```

1 def bw_g(E, N, M, exact=True):
2     return sum((-1)**k * special.comb(N, k, exact=exact) * special.comb(E + N - 1 - k*(M + 1), E - k*(M +
   ↪ 1), exact=exact)
3         for k in range(int(E / M) + 1))
4 def exact_bw_gs(N, M):
5     Es = np.arange(N*M + 1)
6     gs = np.vectorize(bw_g)(np.arange(1 + N*M // 2), N, M, exact=False)
7     return Es, reflect(gs, len(Es) % 2 == 1)

1 def gray_g(E, N, M, exact=True):
2     return 2 * bw_g(E, N, M, exact=exact)
3 def exact_gray_gs(N, M):
4     Es = np.arange(N*M + 1)
5     gs = np.vectorize(gray_g)(np.arange(1 + N*M // 2), N, M, exact=False)
6     return Es, reflect(gs, len(Es) % 2 == 1)

```

Expected results for black/white and gray.

```

1 bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
2 gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)

```

Choose what to compare to.

```

1 Es, gs = bw_Es, bw_gs

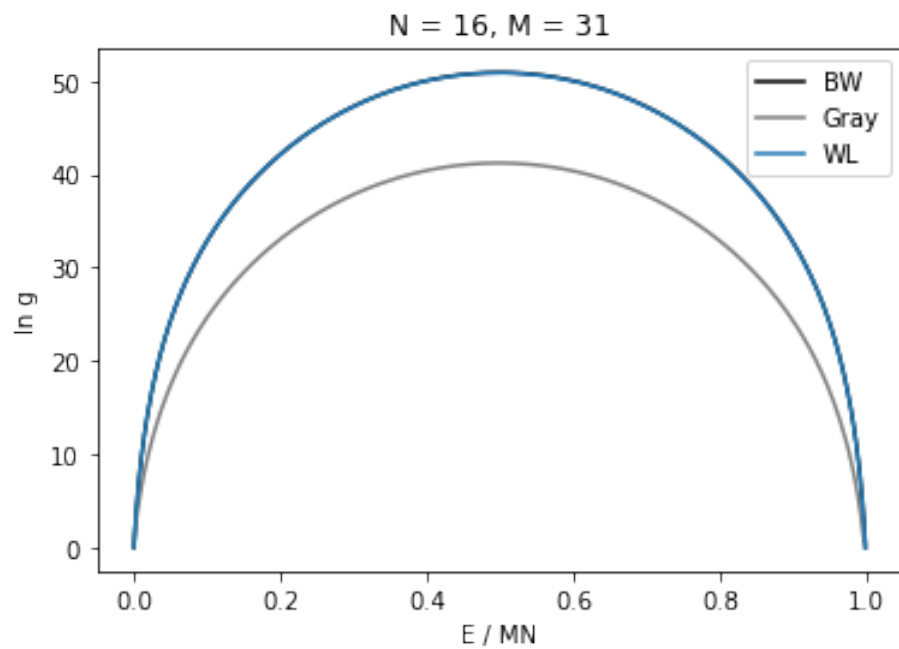
```

Presumably all of the densities of states for different images fall in the region between the all-gray and all-black/white curves.

```

1 plt.plot(bw_Es / len(bw_Es), np.log(bw_gs), 'black', label='BW')
2 plt.plot(gray_Es / len(gray_Es), np.log(gray_gs), 'gray', label='Gray')
3 plt.plot(wl_Es / len(wl_Es), S - min(S), label='WL')
4 plt.xlabel('E / MN')
5 plt.ylabel('ln g')
6 plt.title('N = {}, M = {}'.format(N, M))
7 plt.legend();

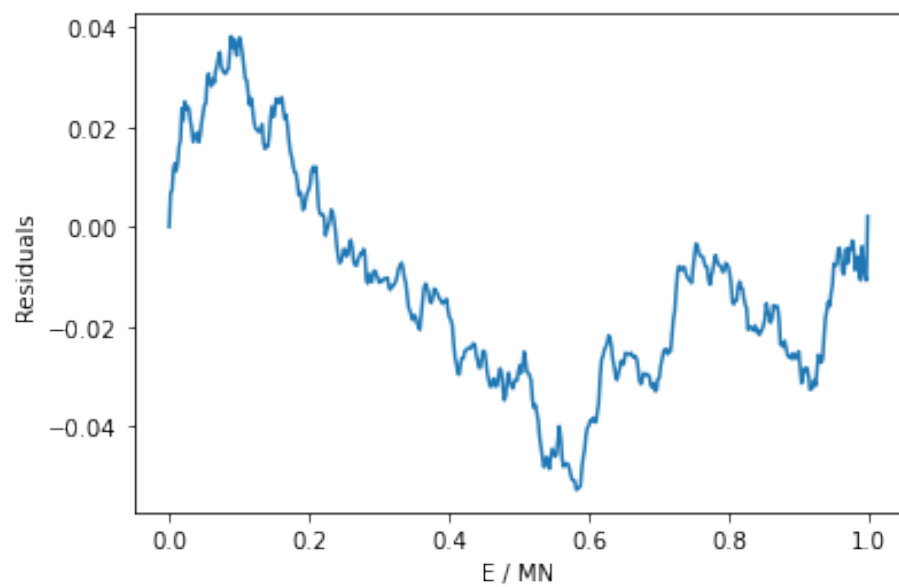
```



```

1 # plt.plot(wlEs / len(wlEs), np.abs(wlgs - bw_gs) / bw_gs)
2 # plt.ylabel('Relative error')
3 plt.plot(wlEs / len(wlEs), S - np.log(bw_gs) - min(S))
4 plt.ylabel('Residuals')
5 plt.xlabel('E / MN');

```



```
1 print('End of job.')
```

End of job.