0.1 Thermal calculations on images

```python
import numpy as np
from scipy import interpolate, special
```

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

0.1.1 Parallel Simulation

```python
N = 16
M = 2**5 - 1
I0 = np.zeros(N, dtype=int)
system_params = {
    'StatisticalImage': {
        'I0': I0,
        'I': I0.copy(),
        'M': M
    }
}
```

```python
# L = 16
# system_params = {
#     'IsingModel': {
#         'spins': np.ones((L, L), dtype=int)
#     }
# }
```

```python
params = {
    'system': system_params,
    'simulation': {
        'max_sweeps': 500_000_000,
        'flat_sweeps': 10_000,
        'eps': 1e-8,
        'logf0': 1,
        'flatness': 0.1
    },
    'parallel': {
        'bins': 8,
        'overlap': 0.25,
        'sweeps': 1_000_000
    },
    'save': {
        'prefix': 'simulation-',
        'dir': 'data'
    }
}
```

```python
params.pop('parallel', None) # Single run
wlresults = 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
```

```python
[r['converged'] for r in wlresults['results']]
```

```
[True]
```

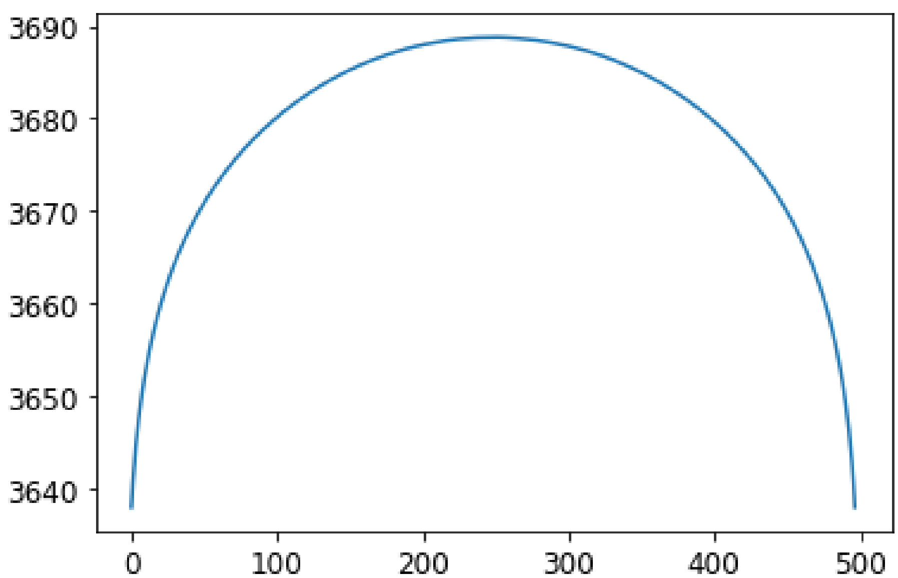
0.1.2 Results

```python
import matplotlib.pyplot as plt
plt.rcParams['font.size'] = 12
```

```python
import h5py, hickle
with h5py.File('data/simulation-gozi5xqv.h5', 'r') as f:
    wlresults = hickle.load(f)
system_params = wlresults['parameters']['system']
```

```python
wlEs, S, ΔS = wl.join_results([wlresults['results']])
```

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




```python
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.

```python
gspl = interpolate.splrep(wlEs, S, s=0*np.sqrt(2))
wlgs = np.exp(interpolate.splev(wlEs, 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.

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

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

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

Expected results for black/white and gray.

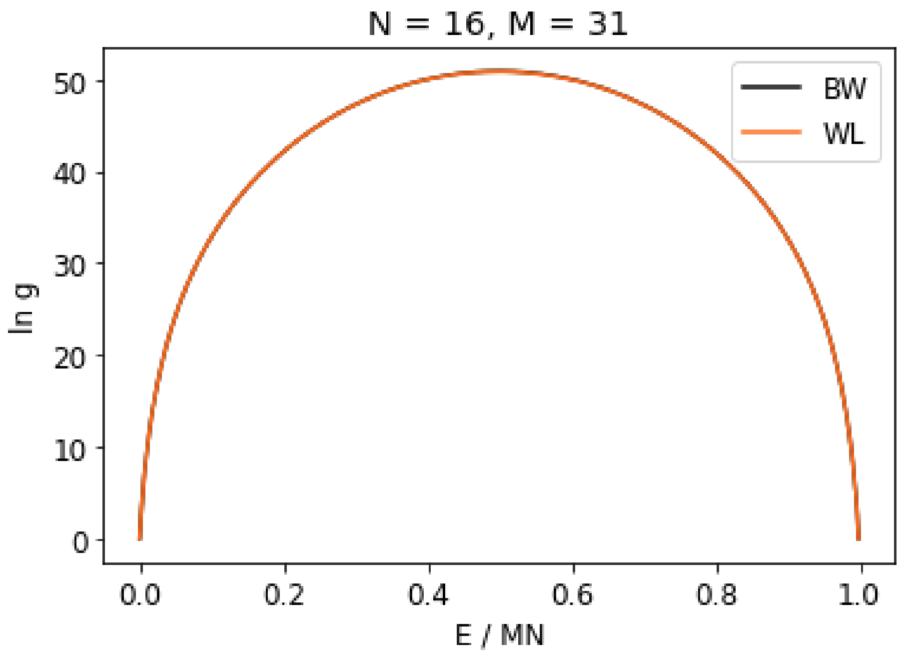
```
bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)
```

Choose what to compare to.

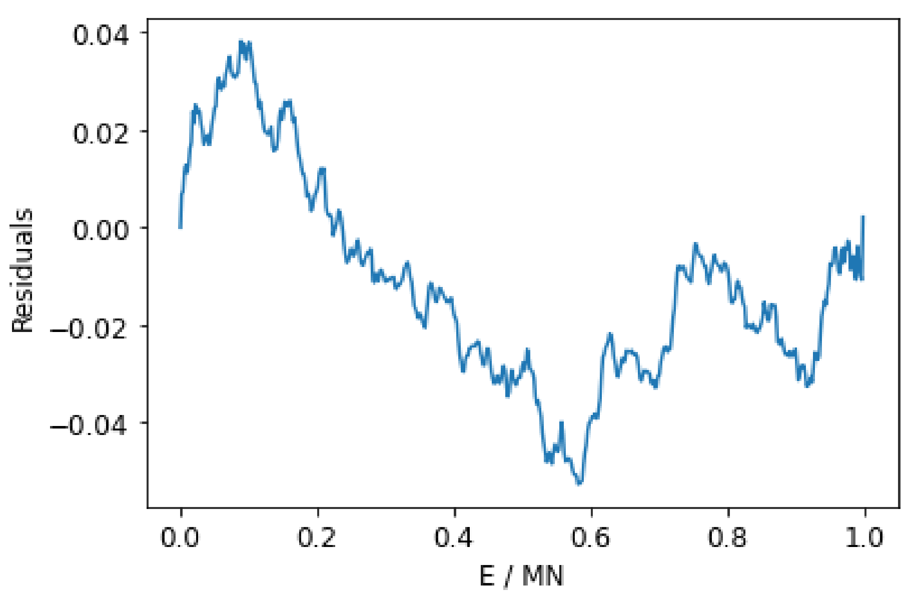
```
Es, gs = bw_Es, bw_gs
```

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

```
plt.plot(bw_Es / len(bw_Es), np.log(bw_gs), 'black', label='BW')
plt.plot(wlEs / len(wlEs), S - min(S), '#ff6716', label='WL')
plt.xlabel('E / MN')
plt.ylabel('ln g')
plt.title('N = {}, M = {}'.format(N, M))
plt.legend()
plt.savefig('wanglandau-bw.png', dpi=600)
```




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



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

```
End of job.
```