0.1 Thermal calculations on images

```python
from numba import njit
from numba.experimental import jitclass
from numba import int64
integer = int64
```

```python
import numpy as np
from scipy import interpolate, special
import os
import tempfile
import h5py, hickle
import pprint
```

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

0.1.1 Parallel Simulation

```python
N = 16
M = 2**5 - 1
Moff = 0
I0 = Moff * np.ones(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': 10_000_000,
        'flat_sweeps': 10_000,
        'eps': 1e-9,
        'logf0': 1,
        'flatness': 0.1
    },
    'parallel': {
        'bins': 2,
        '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': 10000000,
                'flat_sweeps': 10000,
                'eps': 1e-09,
                'logf0': 1,
                'flatness': 0.1},
 'save': {'prefix': 'simulation-', 'dir': 'data'},
 'log': True}

Running ...
Wang-Landau START
fiter    steps       max steps
-----    -----       ---------
1    1120000     97044906
2    480000      124608126
3    800000      141199505
4    800000      150306090
5    1280000     155077318
6    1600000     157519430
7    2080000     158754871
8    3360000     159376220
9    3360000     159687805
10   4320000     159843827
11   5280000     159921895
12   10400000    159960943
13   19680000    159980470
14   17120000    159990235
15   28640000    159995118
16   17440000    159997559
17   98880000    159998780
18   50240000    159999390
19   35840000    159999695
20   159999848   159999848
21   36320000    159999924
22   88960000    159999962
23   63680000    159999981
24   38880000    159999991
25   30880000    159999996
26   125920000   159999998
```

```
27   90560000    159999999
28   68800000    160000000
29   19200000    160000000
30   26720000    160000000
31   57920000    160000000
Done:  1110559848  total MC iterations; not converged.
... done in 122 seconds.
Writing results ... done:  data/simulation-v7qs5a48.h5
```

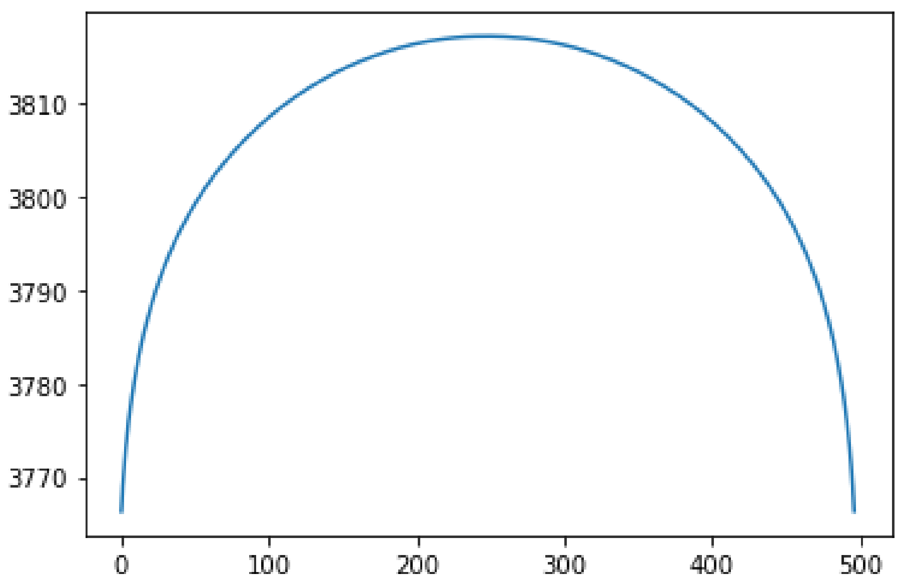
```python
wlEs, S, ΔS = wl.join_results(wlresults['results'])
```

0.1.2 Results

```python
import matplotlib.pyplot as plt
```

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

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



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 solution

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

```python
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.

```python
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)
```

```python
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.

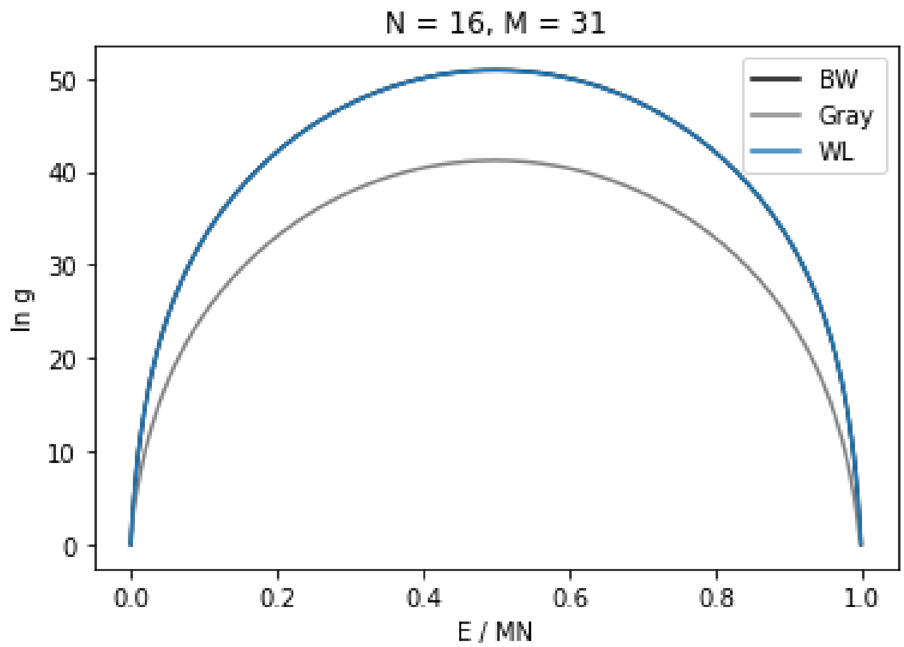
```python
bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)
```

Choose what to compare to.

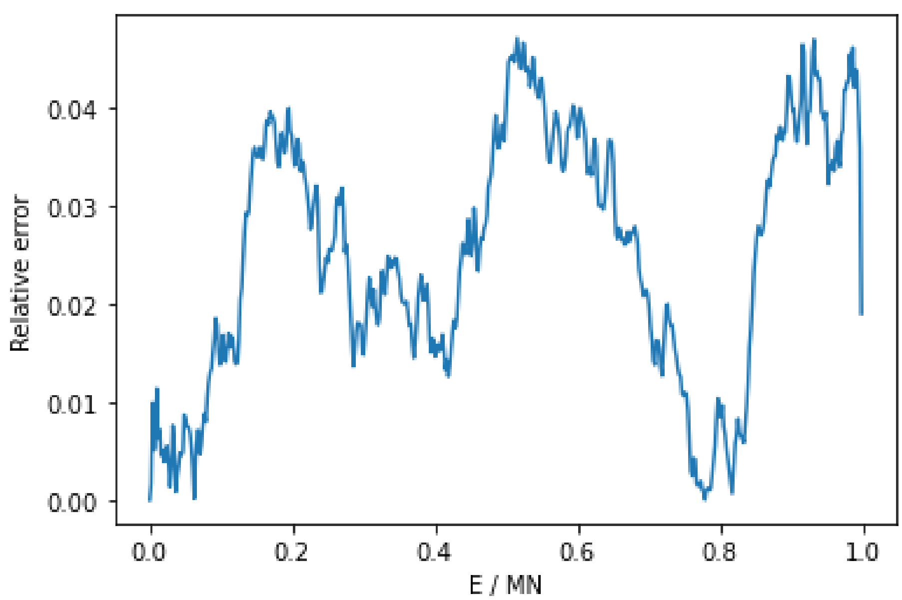
```python
Es, gs = bw_Es, bw_gs
```

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

```python
plt.plot(bw_Es / len(bw_Es), np.log(bw_gs), 'black', label='BW')
plt.plot(gray_Es / len(gray_Es), np.log(gray_gs), 'gray', label='Gray')
plt.plot(wlEs / len(wlEs), S - min(S), label='WL')
plt.xlabel('E / MN')
plt.ylabel('ln g')
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```




```
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.
```