0.1 Thermal calculations on images

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

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

```
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

```
N = 16
Moff = 0
I0 = Moff * np.ones(N, dtype=int)
system_params = {
    'StatisticalImage': {
        'I0': I0,
        'I': I0.copy(),
        'M': 2**5 - 1
    }
}
```

```
params = {
    'system': system_params,
    'simulation': {
        'M': 1_000_000,
        'eps': 1e-8,
        'logf0': 1,
        'flatness': 0.2
    },
    'parallel': {
        'bins': 2,
        'overlap': 0.25,
        'steps': 1_000_000
    },
    'save': {
```

```
        'prefix': 'simulation-',
        'dir': 'data'
    }
}
```

```
# 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,  7, 16, 23, 18,  3, 10,  5, 17,  6,  3,  8,  2, 20, 10,
                                 'M': 31}},
 'simulation': {'M': 1000000, 'eps': 1e-08, 'logf0': 1, 'flatness': 0.2},
 'parallel': {'bins': 2, 'overlap': 0.25, 'steps': 1000000},
 'save': {'prefix': 'simulation-', 'dir': 'data'},
 'log': True}

Finding parallel bin systems ... done.
Running || (()) || done in 17 seconds.
Writing results ... done:  data/simulation-l3t1t_sn.h5
```

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

```
[(r['steps'], r['converged']) for r in wlresults['results']]
```

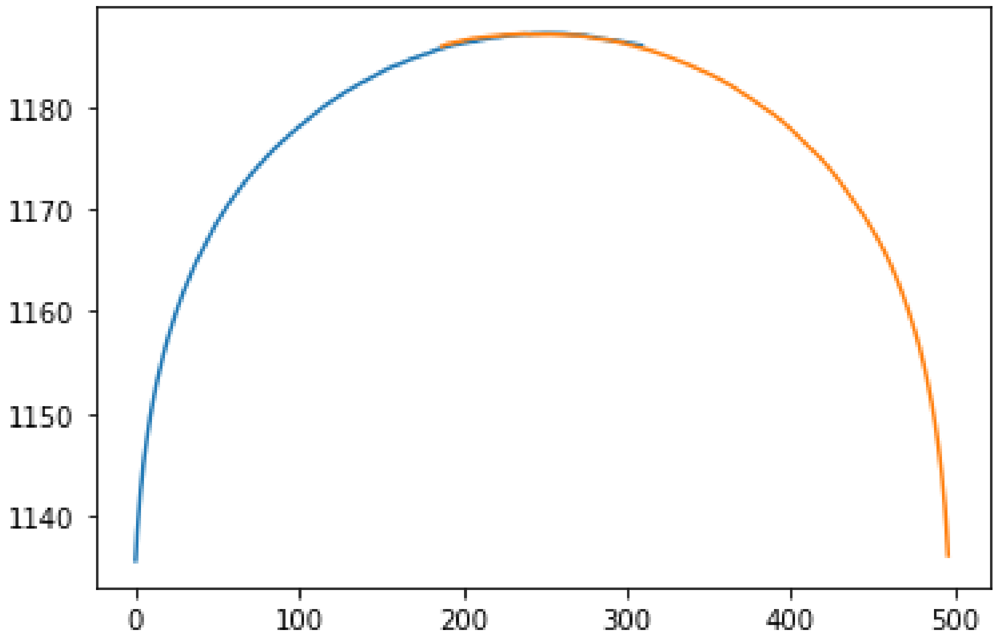
```
[(21351745, False), (21411078, False)]
```

0.1.2 Results

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

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

```
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.

```
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.

```
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.

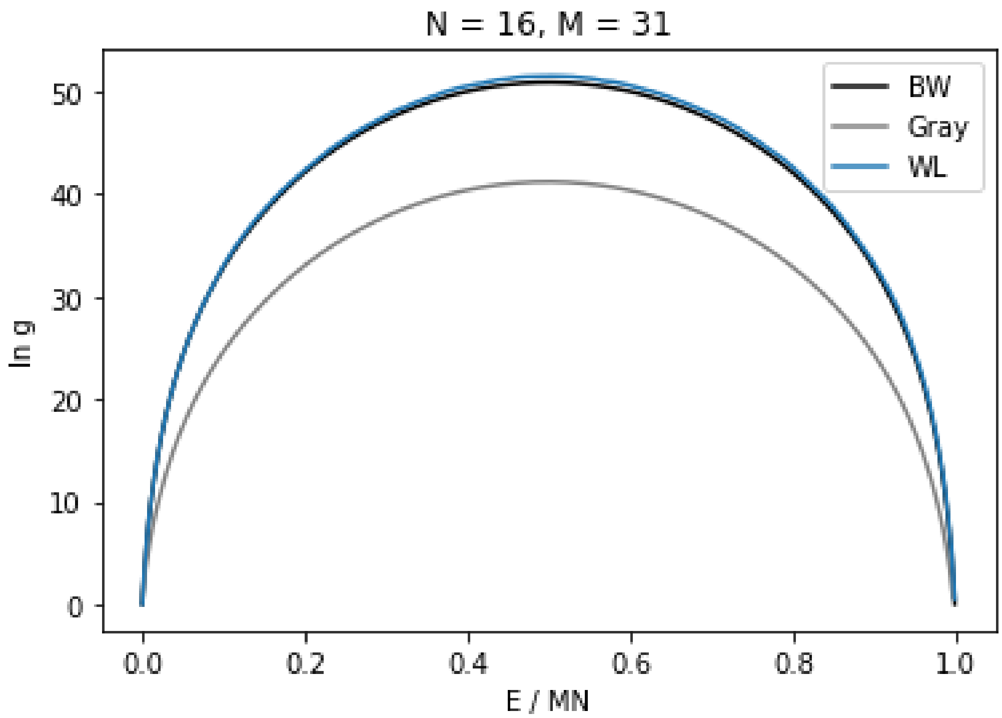
```
bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)
```

Choose what to compare to.

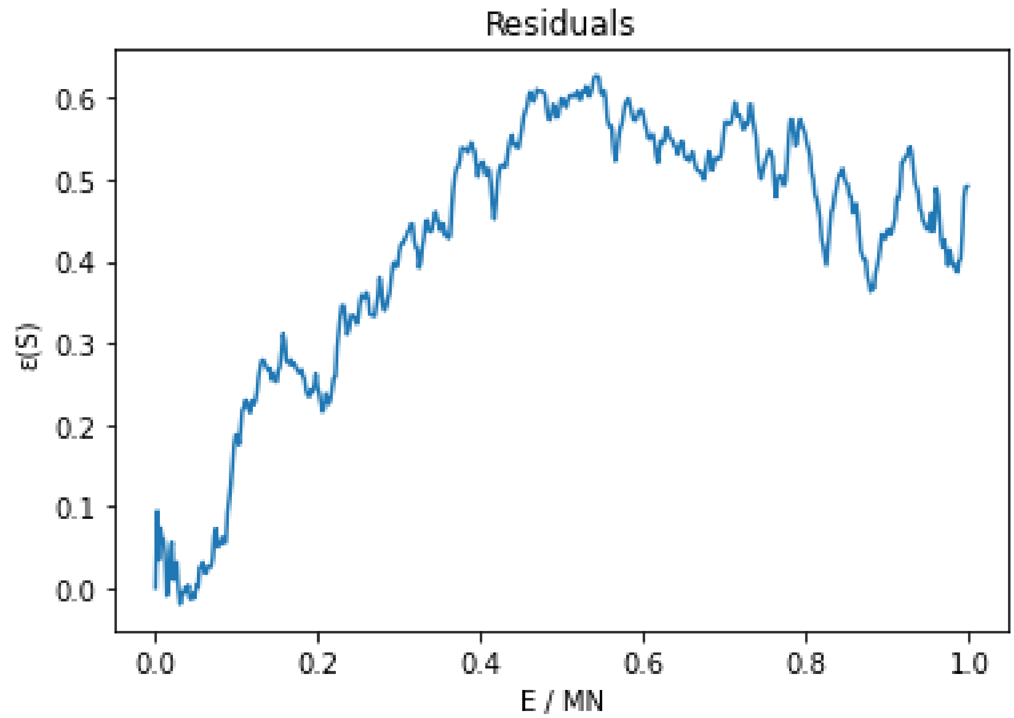
```
Es, gs = bw_Es, bw_gs
```

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(gray_Es / len(gray_Es), np.log(gray_gs), 'gray', label='Gray')
plt.plot(wlEs / len(wlEs), np.log(wlgs), label='WL')
plt.xlabel('E / MN')
plt.ylabel('ln g')
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```




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




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

```
End of job.
```