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
Moff = 0
I0 = Moff * np.ones(N, dtype=int)
system_parameters = {
    'StatisticalImage': {
        'I0': I0,
        'I': I0.copy(),
        'M': 2**5 - 1
    }
}
wl_parameters = {
    'M': 1_000_000,
    'ε': 1e-10,
    'logf0': 1,
    'flatness': 0.1,
    'logging': False
}
parallel_parameters = {
    'bins': 8,
    'overlap': 0.5,
    'steps': 1_000_000,
    'logging': True
}
parameters = {
    'system': system_parameters,
```

```
    'simulation': wl_parameters,
    'parallel': parallel_parameters
}
```

```
print('Run parameters')
print('--------------')
pprint.pp(parameters, sort_dicts=False)
print()
```

```
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': {'M': 1000000,
                'ε': 1e-10,
                'logf0': 1,
                'flatness': 0.1,
                'logging': False},
 'parallel': {'bins': 8, 'overlap': 0.5, 'steps': 1000000, 'logging': True}}
```

```
psystems = sim.make_psystems(wl.parallel_systems, parameters)
```

```
Finding parallel bin systems ... done.
```

```
wlresults = sim.run_parallel(wl.simulation, psystems, parameters)
```

```
Running | (((((((()))))))) | done in 22 seconds.
```

```
sEs, sS = sim.join_results([(Es, S) for Es, S, _ in wlresults])
```

```
with tempfile.NamedTemporaryFile(mode='wb', prefix='wlresults-image-', suffix='.hdf5',
↪  dir='data', delete=False) as f:
    with h5py.File(f, 'w') as hkl:
        print('Writing results ... ', end='', flush=True)
        hickle.dump({
            'parameters': parameters,
            'results': {
                'composite': {
                    'Es': sEs,
                    'S': sS
                },
                'parallel': wlresults # make dict of Es, S, H?
            },
        }, hkl)
        print('done: {}'.format(os.path.relpath(f.name)))
```

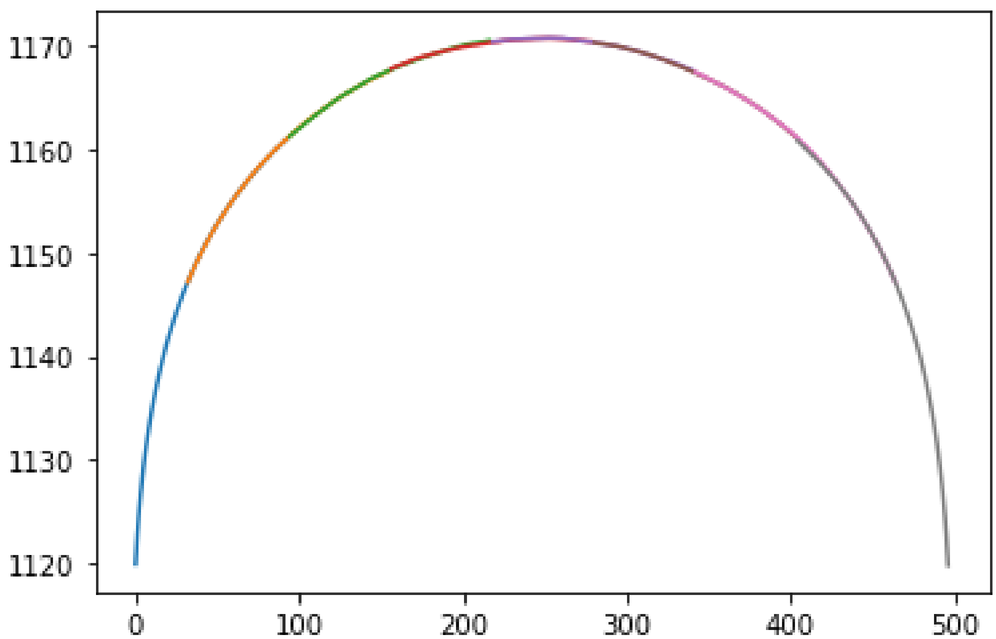
```
Writing results ... done: data/wlresults-image-9380kqhk.hdf5
```

0.1.2 Results

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

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

```
for Es, S, H in wlresults:
    plt.plot(Es[:-1], S)
```




```
wlEs, S = sEs[:-1], sS
```

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.

```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.

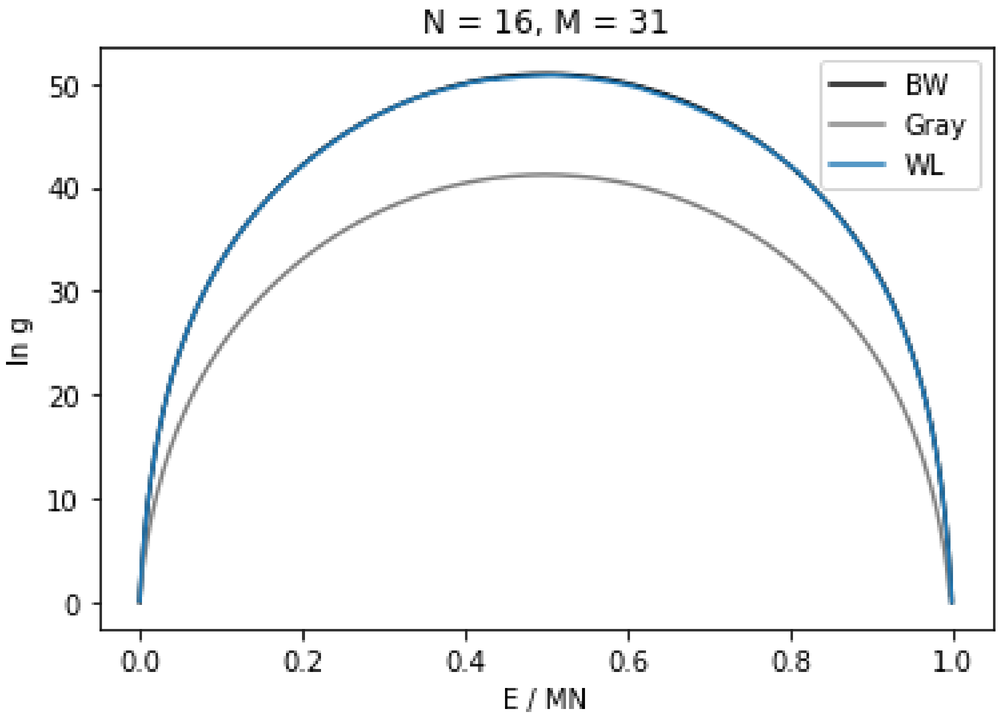
```python
bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)
```

Choose what to compare to.

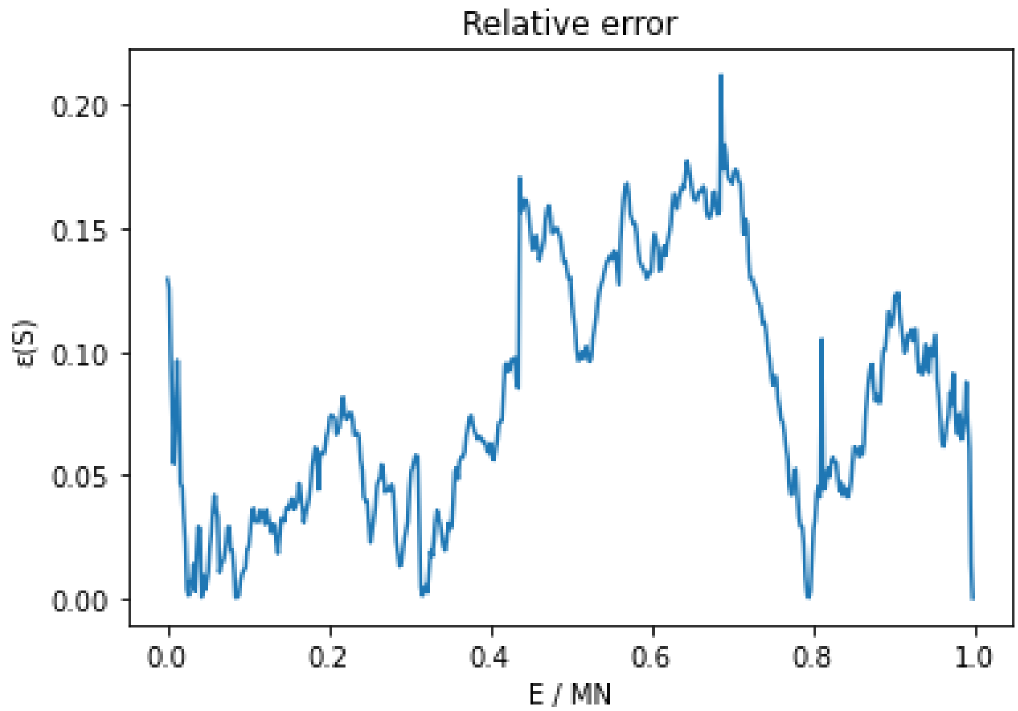
```python
Es, gs = bw_Es, bw_gs
```

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), np.log(wlgs), 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.title('Relative error')
plt.xlabel('E / MN')
plt.ylabel('ε(S)');
```




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

```
End of job.
```