0.1 Thermal calculations on images

```python
from numba import jit
from numba.experimental import jitclass
from numba import int64
```

```python
import numpy as np
from scipy import interpolate, special
import os
import tempfile
import h5py, hickle
from multiprocessing import Pool
from pprint import pprint # for logging
```

```python
# We extend the path instead of using `src.module` to be able to run generated files.
import sys
if 'src' not in sys.path: sys.path.append('src')
import wanglandau as wl
```

```python
integer = int64
spec = [
    ('I0', integer[:]),
    ('I',  integer[:]),
    ('N',  integer),
    ('M',  integer),
    ('E',  integer),
    ('Ev', integer),
    ('dE', integer),
    ('dx', integer),
    ('i',  integer)
]

@jit(nopython=True)
def from_state(state):
    I0, M, I = state
    s = StatisticalImage(I0, M)
    s.I = I.copy()
    s.E = s.energy()
    s.Ev = s.E
    return s

@jitclass(spec)
class StatisticalImage:
    def __init__(self, I0, M = 2**8 - 1):
        self.I0 = I0
        self.I  = I0.copy()
```

```python
        self.N  = len(I0)
        self.M  = M
        self.E  = self.energy()
        self.Ev = self.E
        self.dE = 0
        self.dx = 0
        self.i  = 0
    def state(self):
        return self.I0, self.M, self.I
    def copy(self):
        return from_state(self.state())
    def energy(self):
        return np.sum(np.abs(self.I - self.I0))
    def propose(self):
        i = np.random.randint(self.N)
        self.i = i
        x0 = self.I0[i]
        x = self.I[i]
        r = np.random.randint(2)
        if x == 0:
            dx = r
        elif x == self.M:
            dx = -r
        else:
            dx = 2*r - 1
        dE = np.abs(dx) if x0 == x else (dx if x0 < x else -dx)
        self.dx = dx
        self.dE = dE
        self.Ev = self.E + dE
    def accept(self):
        self.I[self.i] += self.dx
        self.E = self.Ev
```

```python
def parameters(system):
    params = ['N', 'M', 'I0']
    for param in params:
        print(param, "\t", system.__getattribute__(param))
```

0.1.1 Parallel Simulation

```python
N = 16
M = 2**5 - 1
Moff = 0

nsystems = 8
```

```python
overlap = 0.5
wlM = N*M * 100_000_000

system = StatisticalImage(Moff * np.ones(N, dtype=int), M) # Intermediate value
Es = np.arange(N*(M - Moff) + 1 + 1) # for Moff < M / 2
```

```python
print('Parallel Wang-Landau simulation with')
print('\tStep scale {}'.format(wlM))
print('\tParallels  {}'.format(nsystems))
print('\tOverlap    {}'.format(overlap))
print('on a {} with parameters'.format(system.__class__.__name__))
parameters(system)
```

```
Parallel Wang-Landau simulation with
    Step scale 49600000000
    Parallels  8
    Overlap    0.5
on a StatisticalImage with parameters
N    16
M    31
I0   [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

```python
def parallel_wanglandau(subsystem): # Convenient form for `Pool.map`
    wl.urandom_reseed()
    state, Es = subsystem
    system = from_state(state)
    results = wl.wanglandau(system, Es, M = wlM, ε = 1e-16, logging=False)
    print('*', end='', flush=True)
    return results
```

```python
print('Finding parallel bin systems ... ', end='', flush=True)
psystems = wl.parallel_systems(system, Es, n = nsystems, k = overlap, N = 1_00_000)
print('done.')
```

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

```python
print('Running | ', end='', flush=True)
with Pool() as pool:
    wlresults = pool.map(parallel_wanglandau, psystems)
print(' | done.')
```

```
Running | ******** | done.
```

```python
sEs, sS = wl.stitch_results(wlresults)
```

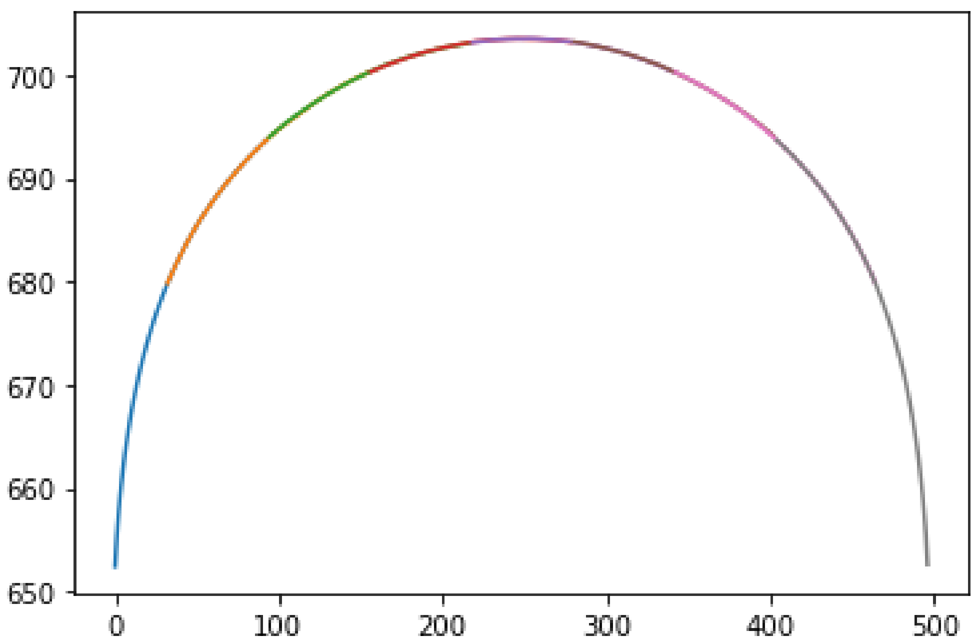
```
with tempfile.NamedTemporaryFile(mode='wb', prefix='wlresults-image-', suffix='.hdf5', dir='data', delete=False) as f:
    with h5py.File(f, 'w') as hkl:
        print('Writing results ... ', end='', flush=True)
        hickle.dump({'N': N, 'M': M, 'wlresults': wlresults, 'sEs': sEs, 'sS': sS}, hkl)
        print('done: {}'.format(os.path.relpath(f.name)))
```

```
Writing results ... done: data/wlresults-image-6_4jpbol.hdf5
```

0.1.2 Results

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

```
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.

```
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.

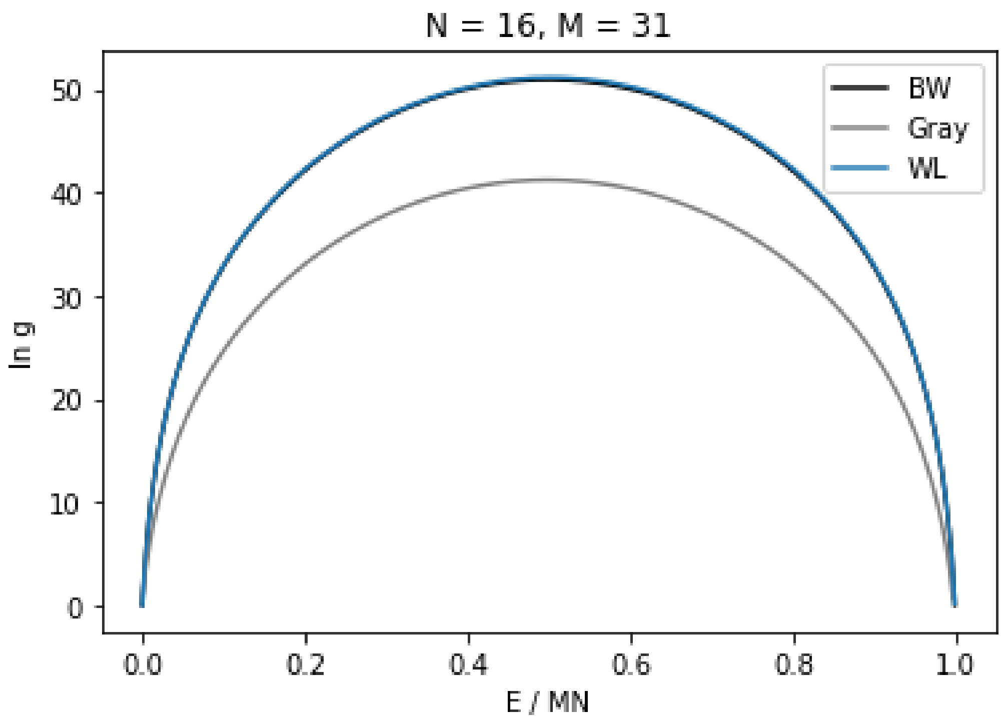
```
bw_Es, bw_gs = exact_bw_gs(N=N, M=M)
gray_Es, gray_gs = exact_gray_gs(N=N, M=-1 + (M + 1) // 2)
```

Choose what to compare to.

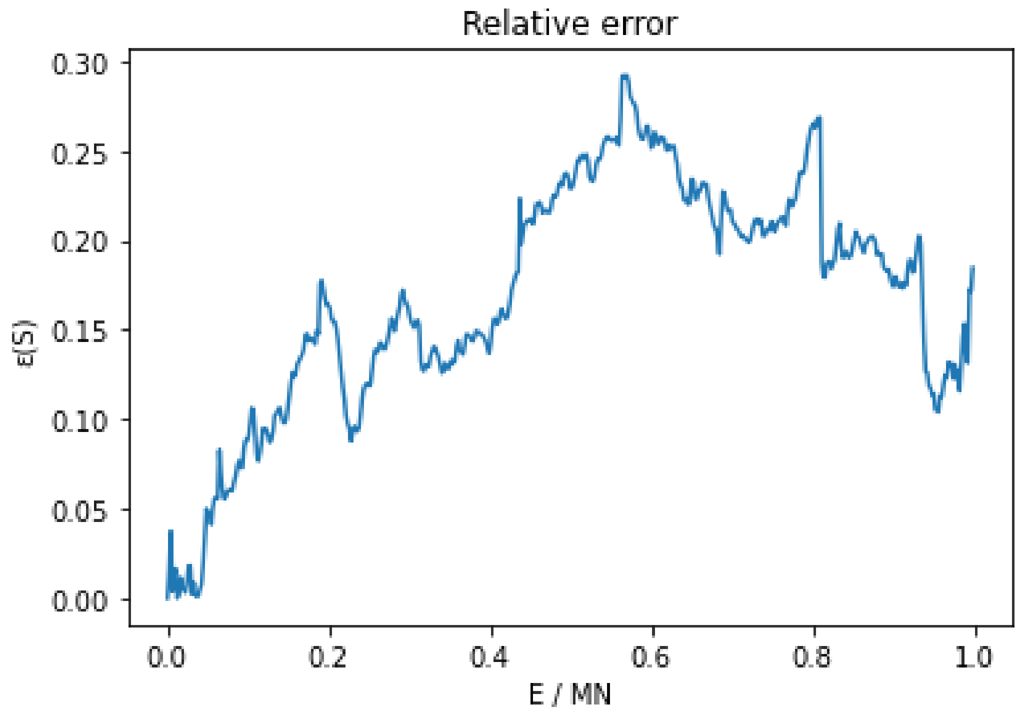
```
Es, gs = bw_Es, bw_gs
```

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

```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();
```




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




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

```
End of job.
```