0.1 Comparison of Wang-Landau results for random Statistical Images

```python
import numpy as np
from scipy import interpolate, special
import os, h5py, hickle
import matplotlib.pyplot as plt
import pprint
plt.rcParams['font.size'] = 12
```

```python
import sys
if 'src' not in sys.path: sys.path.append('src')
import wanglandau as wl
from statistical_image import exact_bw_gs
import canonical_ensemble as canonical
from intensity_entropy import intensity_entropy
```

```python
datadir = 'data/random-images'
paths = [os.path.join(datadir, f) for f in os.listdir(datadir)]
len(paths)
```

```
1024
```

```python
with h5py.File(paths[0], 'r') as f:
    result = hickle.load(f)
    imp = result['parameters']['system']['StatisticalImage']
    N = len(imp['I0'])
    M = imp['M']
    Es = result['results']['Es'][:-1]
```

```python
pprint.pprint(result['parameters'])
```

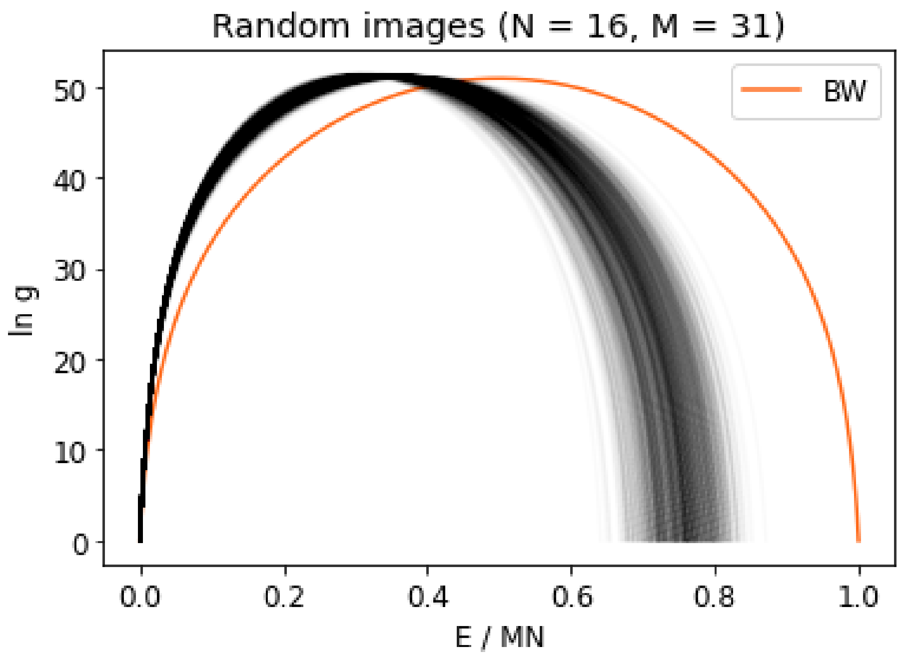
```
{'log': True,
 'simulation': {'eps': 1e-08,
                'flat_sweeps': 10000,
                'flatness': 0.1,
                'logf0': 1,
                'max_sweeps': 100000000},
 'system': {'StatisticalImage': {'I': array([31, 21, 20,  5, 16, 16, 10, 12, 27,  1, 31, 10,  2, 14, 30,  7]),
                                 'I0': array([18, 21, 21,  5, 15, 18,  6, 14, 27,  1, 30, 11,  4, 12, 31,  6]),
                                 'M': 31}}}
```

```python
def file_results(path):
    with h5py.File(path, 'r') as f:
        result = hickle.load(f)
        Es = result['results']['Es'][:-1]
        S = result['results']['S']
        return Es, S - min(S)
```

```python
xEs, xgs = exact_bw_gs(N, M)
xlng = np.log(xgs)
xens = canonical.Ensemble(xEs, xlng, 'Exact')
```

```python
plt.plot(xEs / (N*M), xlng, '#ff6716', label='BW')
for Es, S in map(file_results, paths):
```

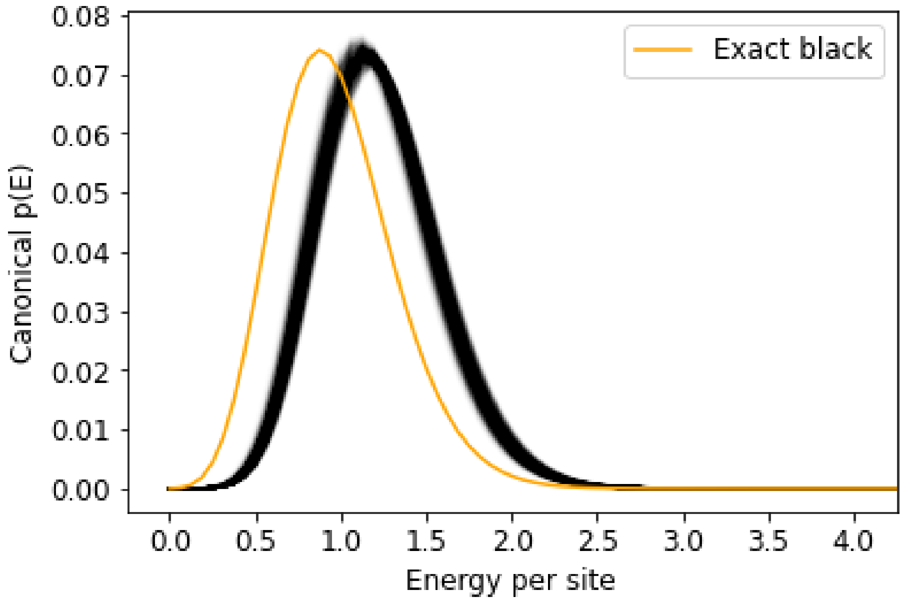
```
    plt.plot(Es / (N*M), S, 'black', alpha=0.02)
plt.title('Random images (N = {}, M = {})'.format(N, M))
plt.xlabel("E / MN")
plt.ylabel("ln g")
plt.legend()
plt.savefig('wanglandau-gray.png', dpi=600)
```




```
βs = np.exp(np.linspace(-8, 4, 500))
βc = 1 / np.sqrt(2)
```

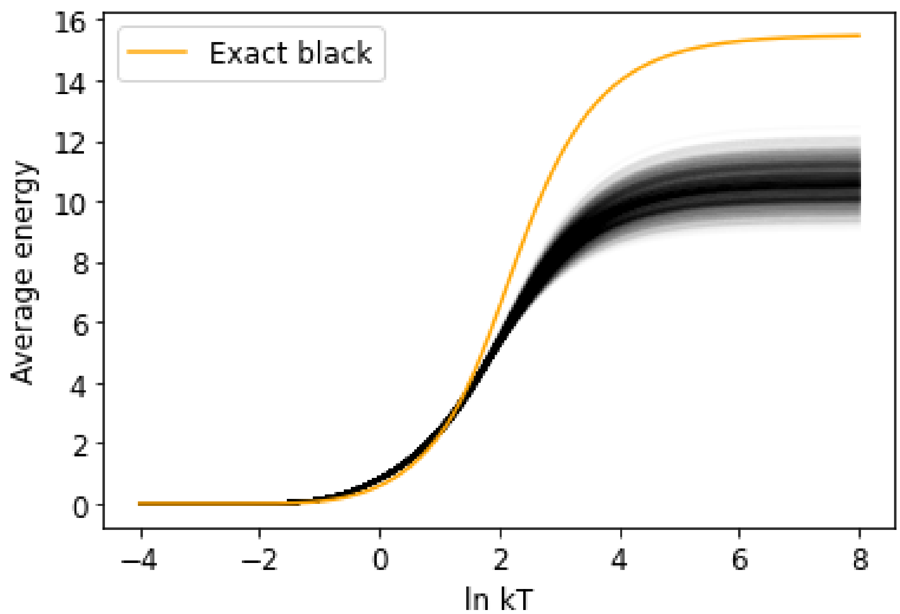
Gibbs distribution

```
plt.xlim(-0.25, 4.25)
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    plt.plot(Es / N, ens.p(βc), 'black', alpha=0.01)
plt.plot(xEs / N, xens.p(βc), 'orange', label='Exact black')
plt.xlabel('Energy per site')
plt.ylabel('Canonical p(E)')
plt.legend();
```



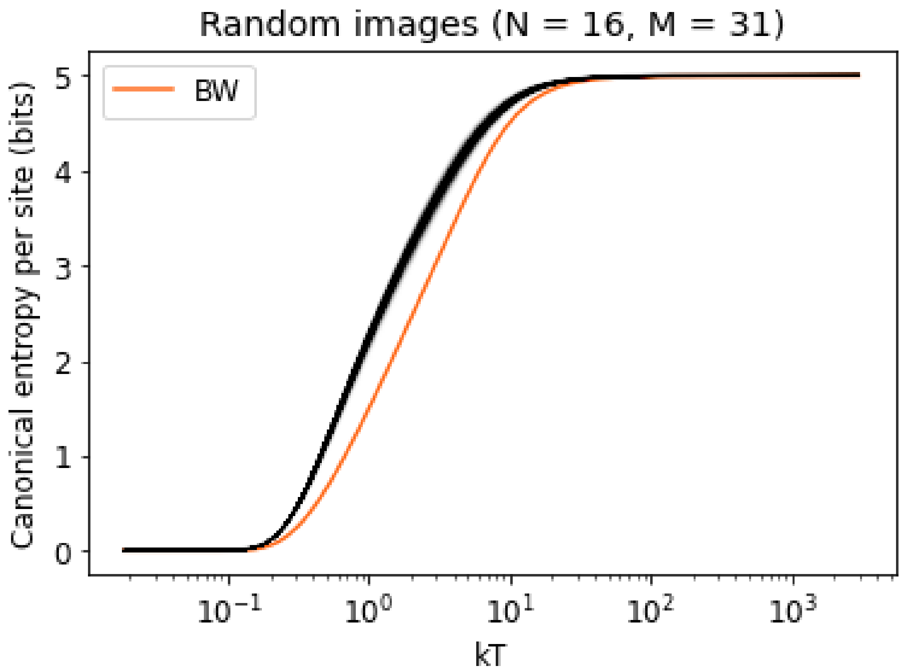
Average energy

```
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    plt.plot(-np.log(βs), ens.energy(βs) / N, 'black', alpha=0.02)
plt.plot(-np.log(βs), xens.energy(βs) / N, 'orange', label='Exact black')
plt.xlabel('ln kT')
plt.ylabel('Average energy')
plt.legend();
```



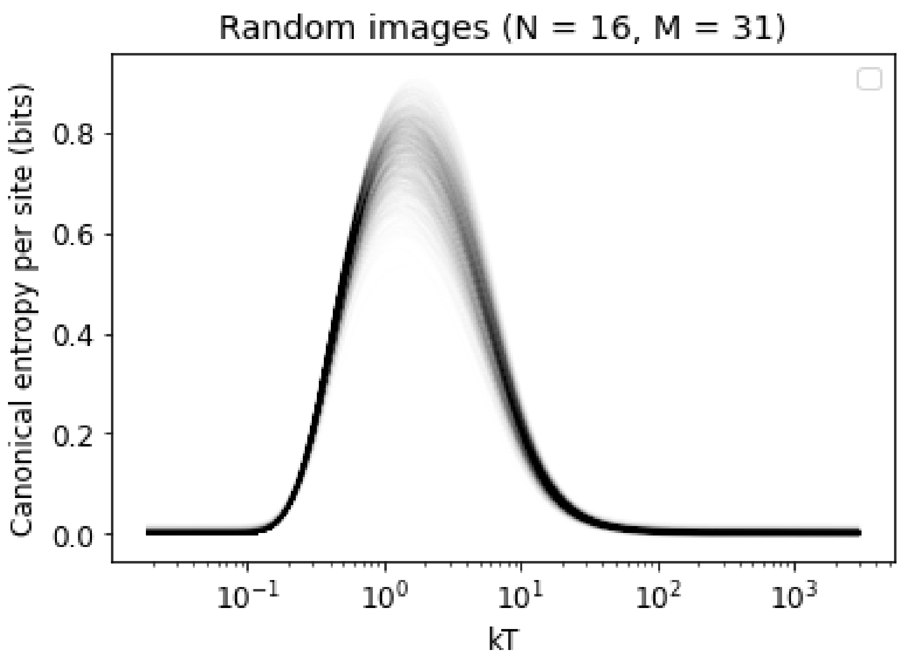
Entropy

```
plt.plot(1 / βs, xens.entropy(βs) / (N*np.log(2)), '#ff6716', label='BW')
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    plt.plot(1 / βs, ens.entropy(βs) / (N*np.log(2)), 'black', alpha=0.002)
plt.title('Random images (N = {}, M = {})'.format(N, M))
plt.xlabel('kT')
plt.xscale('log')
plt.ylabel('Canonical entropy per site (bits)')
plt.legend()
plt.savefig('wanglandau-gray-S.png', dpi=600)
```



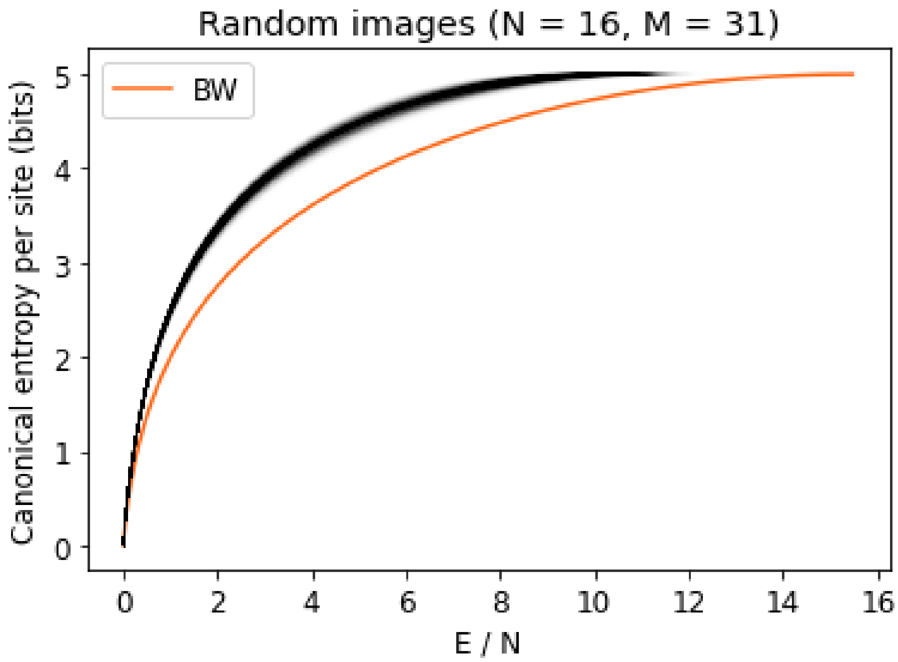

```
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    plt.plot(1 / βs, (ens.entropy(βs) - xens.entropy(βs)) / (N*np.log(2)), 'black', alpha=0.002)
plt.title('Random images (N = {}, M = {})'.format(N, M))
plt.xlabel('kT')
plt.xscale('log')
plt.ylabel('Canonical entropy per site (bits)')
plt.legend();
```

```
No handles with labels found to put in legend.
```



Entropy vs Average energy

```
plt.plot(xens.energy(βs) / N, xens.entropy(βs) / (N*np.log(2)), '#ff6716', label='BW')
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    plt.plot(ens.energy(βs) / N, ens.entropy(βs) / (N*np.log(2)), 'black', alpha=0.002)
plt.title('Random images (N = {}, M = {})'.format(N, M))
plt.xlabel('E / N')
plt.ylabel('Canonical entropy per site (bits)')
plt.legend()
plt.savefig('wanglandau-gray-ES.png', dpi=600)
```



Is the canonical entropy related to the intensity entropy?

```
result['parameters']['system']['StatisticalImage']['I0']
```

```
array([18, 21, 21,  5, 15, 18,  6, 14, 27,  1, 30, 11,  4, 12, 31,  6])
```

```
Sc = σSc = 0
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    Sc += ens.entropy(βc)
Sc /= len(paths)
for Es, S in map(file_results, paths):
    ens = canonical.Ensemble(Es, S)
    σSc += (Sc - ens.entropy(βc))**2
σSc = np.sqrt(σSc / (len(paths) - 1))
```

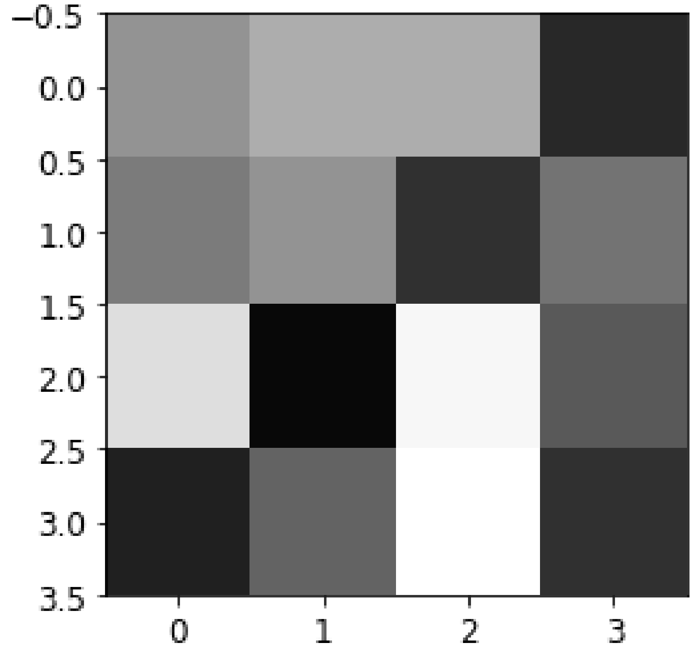
```
Sc / (N * np.log(2))
```

```
array([2.74455418])
```

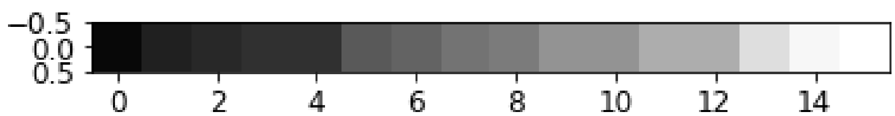
```
σSc / (N * np.log(2))
```

```
array([0.06219699])
```

```
plt.imshow(np.reshape(result['parameters']['system']['StatisticalImage']['I0'], (int(np.sqrt(N)), -1)),
↪  cmap='gray', vmin=0, vmax=M);
```




```
plt.imshow(np.reshape(np.sort(result['parameters']['system']['StatisticalImage']['I0']), (1, -1)),
↪  cmap='gray', vmin=0, vmax=M);
```




```
I0 = result['parameters']['system']['StatisticalImage']['I0']
```

```
intensity_entropy(I0, upper=M+1)
```

```
3.625
```

0.1.1 Metropolis to generate canonical samples if we need them

```
from numba import njit
from statistical_image import StatisticalImage
```

```
si = StatisticalImage(I0, I0.copy(), M)
```

```
@njit
def metropolis_step(β, system, S=1):
    for _ in range(S):
        system.propose()
        E, Ev = system.E, system.Ev
        if Ev ≤ E or np.random.rand() < np.exp(-β*(Ev - E)):
            system.accept()
```