0.1 Calculating canonical ensemble averages

```python
import numpy as np
import matplotlib.pyplot as plt
import os
import h5py, hickle
```

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

```python
class CanonicalEnsemble:
    def __init__(self, Es, lngs, name, λ = None):
        self.Es = Es
        self.lngs = lngs
        self.name = name
        # Choose to scale the exponent of Z to be a convenient size.
        # This can be improved if needed by taking into account typical β*Es.
        self.λ = max(lngs) if λ is None else λ
    def Zλ(self, β):
        return np.sum(np.exp(-(β * self.Es - self.lngs + self.λ)))
    def Z(self, β):
        return np.exp(self.λ) * self.Zλ(β)
    def average(self, f, β):
#         return np.sum(f(self) * self.gs * np.exp(-β * self.Es)) / self.Z(β)
        return np.sum(f(self) * np.exp(-(β * self.Es - self.lngs + self.λ))) / self.Zλ(β)
    def energy(self, β):
        return self.average(lambda ens: ens.Es, β)
    def energy2(self, β):
        return self.average(lambda ens: ens.Es**2, β)
    def heat_capacity(self, β):
        return self.energy2(β) - self.energy(β)**2
    def free_energy(self, β):
        return -np.log(self.Z(β)) / β
    def entropy(self, β):
        return β * self.energy(β) + np.log(self.Z(β))
```

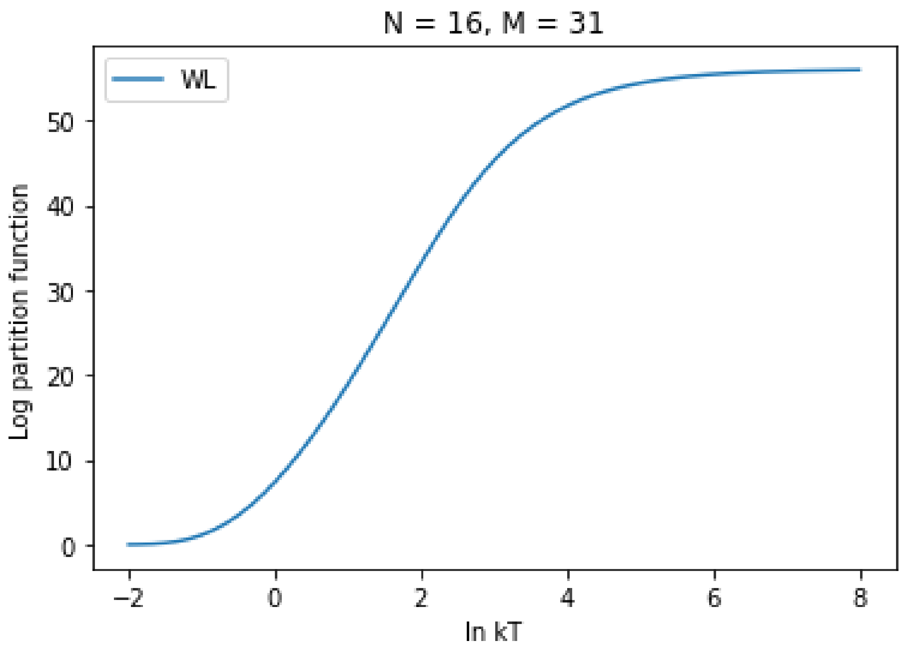
```python
with h5py.File('data/simulation-13t1t_sn.h5', 'r') as f:
    h = hickle.load(f)
    results = h['results']
    params = h['parameters']
```

```python
N, M = len(params['system']['StatisticalImage']['I0']), params['system']['StatisticalImage']['M']
wlEs, S, ΔS = wl.join_results(results)
```

```python
βs = [np.exp(k) for k in np.linspace(-8, 2, 500)]
wlens = CanonicalEnsemble(wlEs, S - min(S), 'WL') # Wang-Landau results
# xens = CanonicalEnsemble(Es, gs, 'Exact') # Exact
# ensembles = [wlens, xens]
ensembles = [wlens]
```

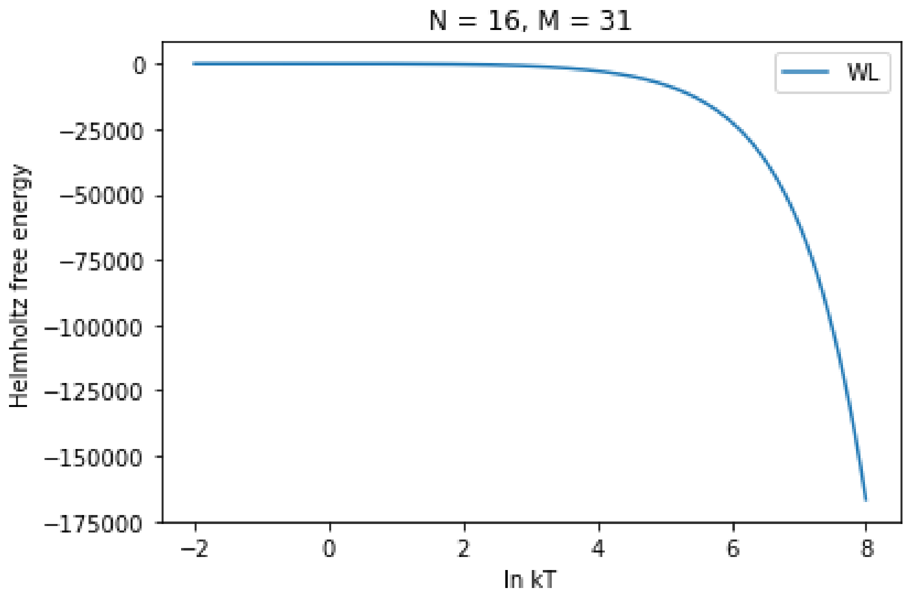
Partition function

```python
for ens in ensembles:
    plt.plot(-np.log(βs), np.log(np.vectorize(ens.Z)(βs)), label=ens.name)
plt.xlabel("ln kT")
plt.ylabel("Log partition function")
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```



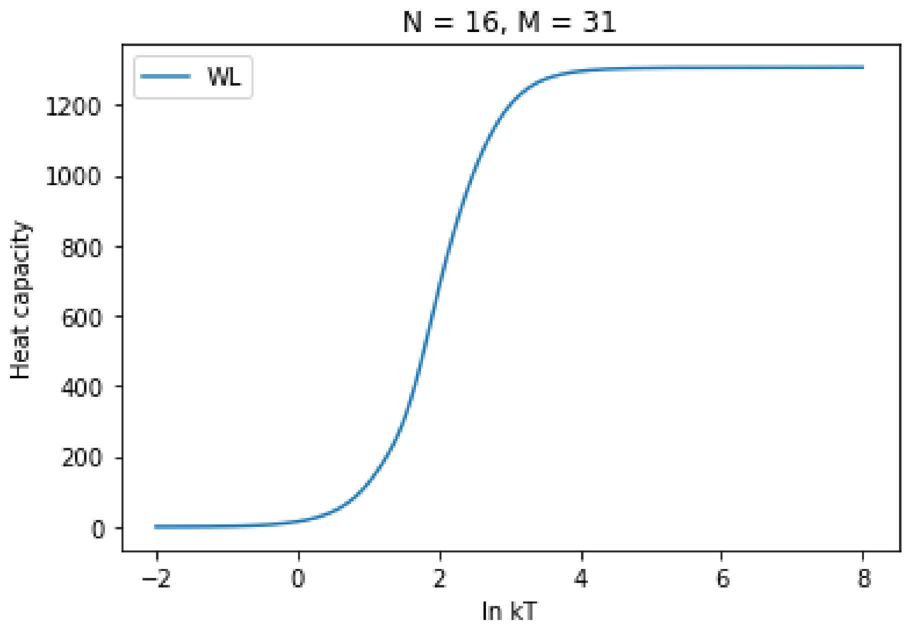
Helmholtz free energy

```
for ens in ensembles:
    plt.plot(-np.log(βs), np.vectorize(ens.free_energy)(βs), label=ens.name)
plt.xlabel("ln kT")
plt.ylabel("Helmholtz free energy")
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```



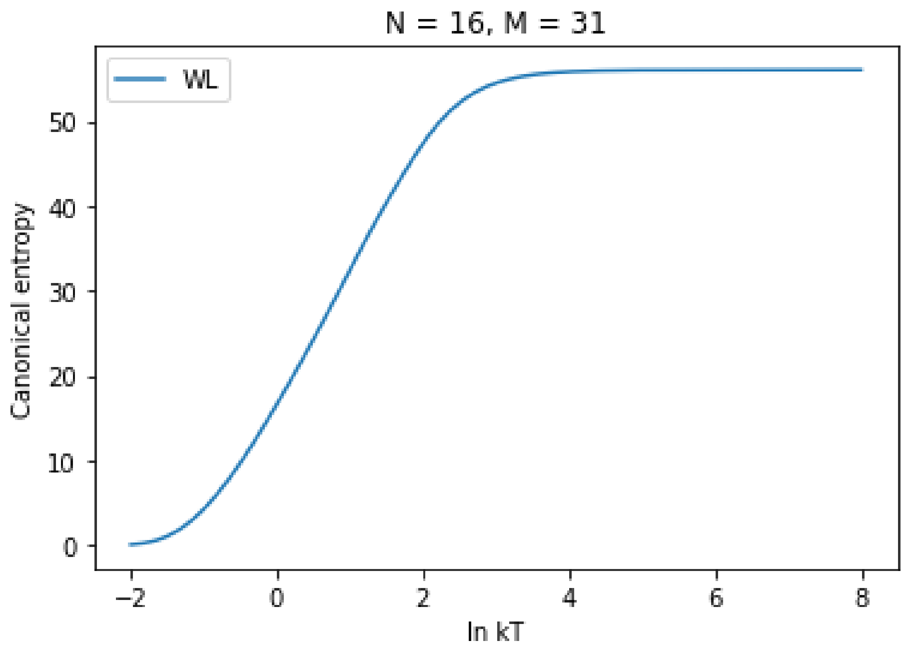
Heat capacity

```
for ens in ensembles:
    plt.plot(-np.log(βs), np.vectorize(ens.heat_capacity)(βs), label=ens.name)
plt.xlabel("ln kT")
plt.ylabel("Heat capacity")
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```



Entropy

```
for ens in ensembles:
    plt.plot(-np.log(βs), np.vectorize(ens.entropy)(βs), label=ens.name)
plt.xlabel("ln kT")
plt.ylabel("Canonical entropy")
plt.title('N = {}, M = {}'.format(N, M))
plt.legend();
```



N = 16, M = 31
WL
Canonical entropy
ln kT
0
10
20
30
40
50
−2
0
2
4
6
8