1 The Wang-Landau algorithm (density of states)

We determine thermodynamic quantities from the partition function by obtaining the density of states from a simulation.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import interpolate
```

The test system is the 2d Ising model.

```
class Ising:
    def __init__(self, n):
        self.n = n
        self.spins = np.sign(np.random.rand(n, n) - 0.5)
        self.E = self.energy()
        self.Ev = self.E
    def neighbors(self, i, j):
        return np.hstack([self.spins[:,j].take([i-1,i+1], mode='wrap'),
                          self.spins[i,:].take([j-1,j+1], mode='wrap')])
    def energy(self):
        return -0.5 * sum(np.sum(s * self.neighbors(i, j))
                          for (i, j), s in np.ndenumerate(self.spins))
    def propose(self):
        i, j = np.random.randint(self.n), np.random.randint(self.n)
        self.i, self.j = i, j
        dE = 2 * np.sum(self.spins[i, j] * self.neighbors(i, j))
        self.dE = dE
        self.Ev = self.E + dE
    def accept(self):
        self.spins[self.i, self.j] *= -1
        self.E = self.Ev
```

Note that this class-based approach adds some overhead. For speed, instances of `Ising` should be inlined into the simulation method.

A Wang-Landau algorithm, with quantities as logarithms and with monte-carlo steps proportional to $f^{-1/2}$ (a "Zhou-Bhat schedule").

```
def flat(H, tol = 0.2):
    """Determines if an evenly-spaced histogram is approximately flat."""
    Hμ = np.mean(H)
    Hf = np.max(H)
    H0 = np.min(H)
    return Hf / (1 + tol) < Hμ < H0 / (1 - tol)
```

```
# def flat(H, tol = 0.2):
#     """Determines if an evenly-spaced histogram is approximately flat."""
#     Hμ = np.mean(H)
#     return not np.any(H < (1 - tol) * Hμ) and np.all(H ≠ 0)
```

```
# Note: some parameters are hardcoded for testing
def density_sim(system):
    randint = np.random.randint
    rand = np.random.rand
    exp = np.exp

    # Parameters
    M = 10_000_000 # Monte carlo step scale
    ε = 1e-8
    logftol = np.log(1 + ε)
    logf0 = 1
    N = 8**2 + 1 # Energy bins
    E0 = -2 * 8**2
    Ef = 2 * 8**2

    ΔE = (Ef - E0) / (N - 1)
    fiters = int(np.ceil(np.log2(logf0) - np.log2(logftol)))
    fiter = 0
    mciters = 0
    Es = np.linspace(E0, Ef, N)
    S = np.zeros(N) # Set all initial g's to 1
    H = np.zeros(N, dtype=int)
    logf = logf0
    # Linearly bin the energy
    i = max(0, min(N - 1, int(round((N - 1) * (system.E - E0) / (Ef - E0)))))
    print("ΔE = {}".format(ΔE))
    while logftol < logf:
        H[:] = 0
        logf /= 2
        iters = 0
        niters = int((M + 1) * exp(-logf / 2))
        fiter += 1
        while not flat(H[2:-2]) and iters < niters: # Ising-specific histogram
#         while not flat(H) and iters < niters:
            system.propose()
            Ev = system.Ev
            j = max(0, min(N - 1, int(round((N - 1) * (Ev - E0) / (Ef - E0)))))
            if E0 - ΔE/2 ≤ Ev ≤ Ef + ΔE/2 and (S[j] < S[i] or rand() < exp(S[i] - S[j])):
                system.accept()
                i = j
            H[i] += 1
```

```
            S[i] += logf
            iters += 1
        mciters += iters
        print("f: {} / {}\t({} / {})".format(fiter, fiters, iters, niters))

    print("Done: {} total MC iterations.".format(mciters))
    return Es, S, H
```

```
isingn = 8
sys = Ising(isingn)
Es, S, H = density_sim(sys);
```

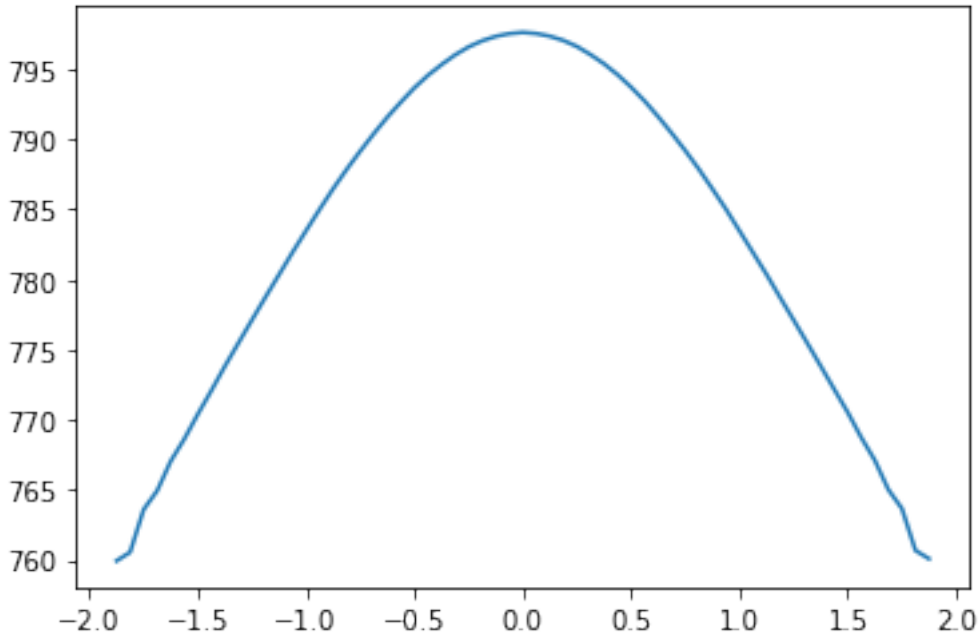
```
ΔE = 4.0
f: 1 / 27   (51914 / 7788008)
f: 2 / 27   (23763 / 8824969)
f: 3 / 27   (28540 / 9394131)
f: 4 / 27   (29971 / 9692333)
f: 5 / 27   (34174 / 9844965)
f: 6 / 27   (47534 / 9922180)
f: 7 / 27   (48944 / 9961014)
f: 8 / 27   (107754 / 9980488)
f: 9 / 27   (179729 / 9990240)
f: 10 / 27  (187907 / 9995119)
f: 11 / 27  (224943 / 9997559)
f: 12 / 27  (1034768 / 9998780)
f: 13 / 27  (244301 / 9999390)
f: 14 / 27  (133628 / 9999695)
f: 15 / 27  (214968 / 9999848)
f: 16 / 27  (1293088 / 9999924)
f: 17 / 27  (420043 / 9999962)
f: 18 / 27  (551351 / 9999981)
f: 19 / 27  (253547 / 9999991)
f: 20 / 27  (394211 / 9999996)
f: 21 / 27  (166352 / 9999998)
f: 22 / 27  (9999999 / 9999999)
f: 23 / 27  (467290 / 10000000)
f: 24 / 27  (213657 / 10000000)
f: 25 / 27  (366069 / 10000000)
f: 26 / 27  (563593 / 10000000)
f: 27 / 27  (126259 / 10000000)
```

```
Done: 17408297 total MC iterations.
```

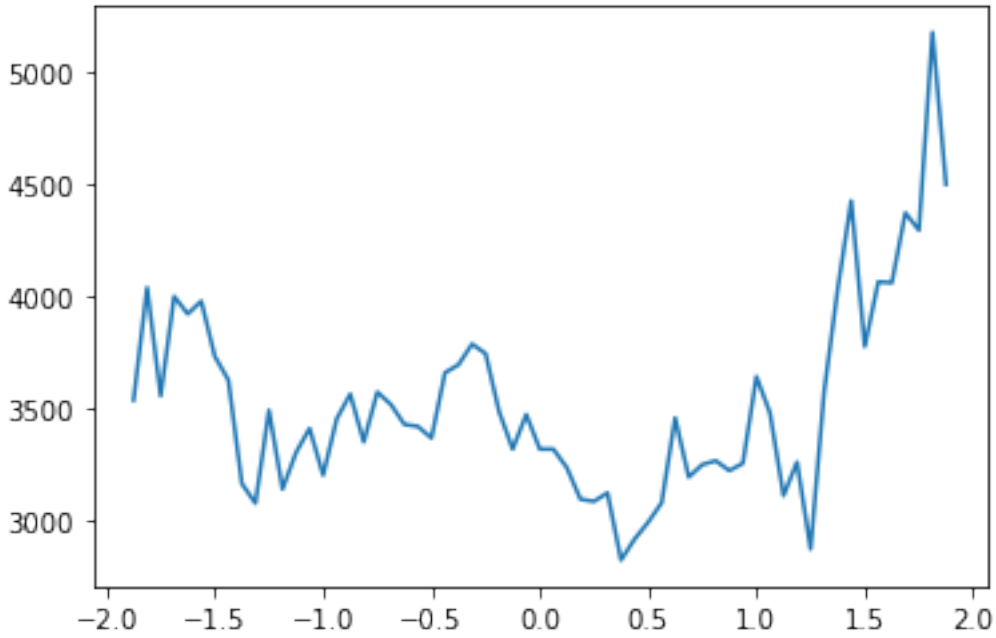
```
H
```

```
array([4027,    0, 3534, 4035, 3553, 3996, 3918, 3975, 3729, 3624, 3162,
       3075, 3489, 3138, 3301, 3408, 3200, 3450, 3561, 3348, 3570, 3515,
       3426, 3417, 3365, 3656, 3692, 3784, 3741, 3482, 3316, 3470, 3316,
       3316, 3233, 3093, 3083, 3121, 2822, 2915, 2990, 3077, 3455, 3191,
       3247, 3264, 3220, 3253, 3639, 3477, 3109, 3258, 2871, 3571, 4028,
       4422, 3773, 4061, 4057, 4368, 4291, 5172, 4495,    0, 6085])
```

```
plt.plot(Es[2:-2] / isingn**2, S[2:-2]);
```




```
plt.plot(Es[2:-2] / isingn**2, H[2:-2]);
```

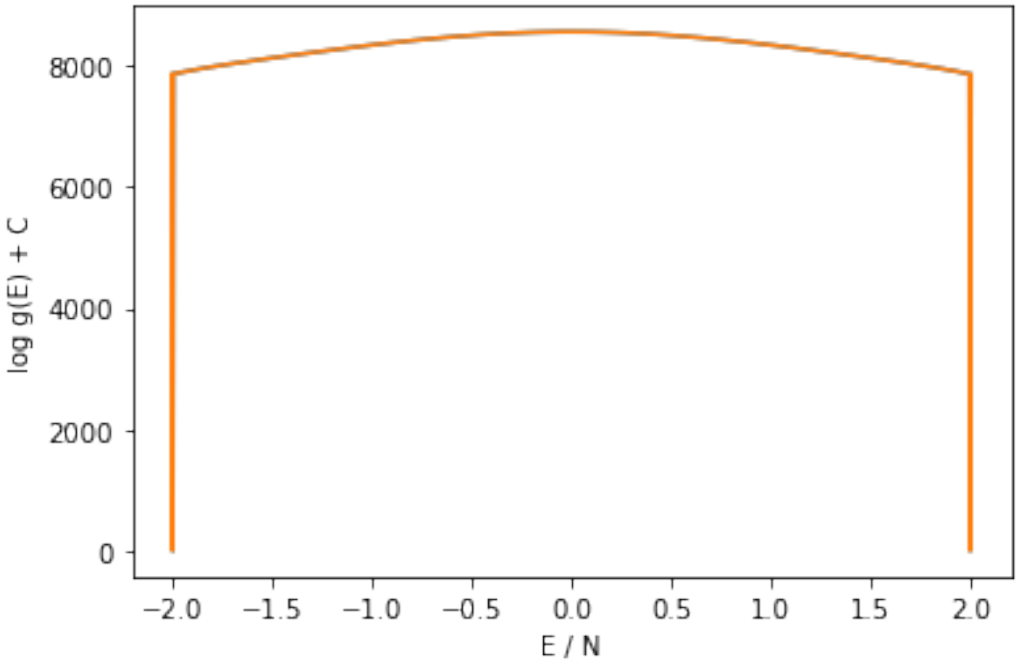


1.1 Calculating canonical ensemble averages

```
gspl = interpolate.splrep(Es, S, s=2*np.sqrt(2))
gs = np.exp(interpolate.splev(Es, gspl) - min(S))
```

```
<ipython-input-414-d6b1add1a212>:2: RuntimeWarning: overflow encountered in exp
  gs = np.exp(interpolate.splev(Es, gspl) - min(S))
```

```
plt.plot(Es / isingn**2, S)
plt.plot(Es / isingn**2, interpolate.splev(Es, gspl))
plt.xlabel("E / N")
plt.ylabel("log g(E) + C");
```



Translate energies to have minimum zero so that Z is representable.

```
nEs = Es - min(Es)
```

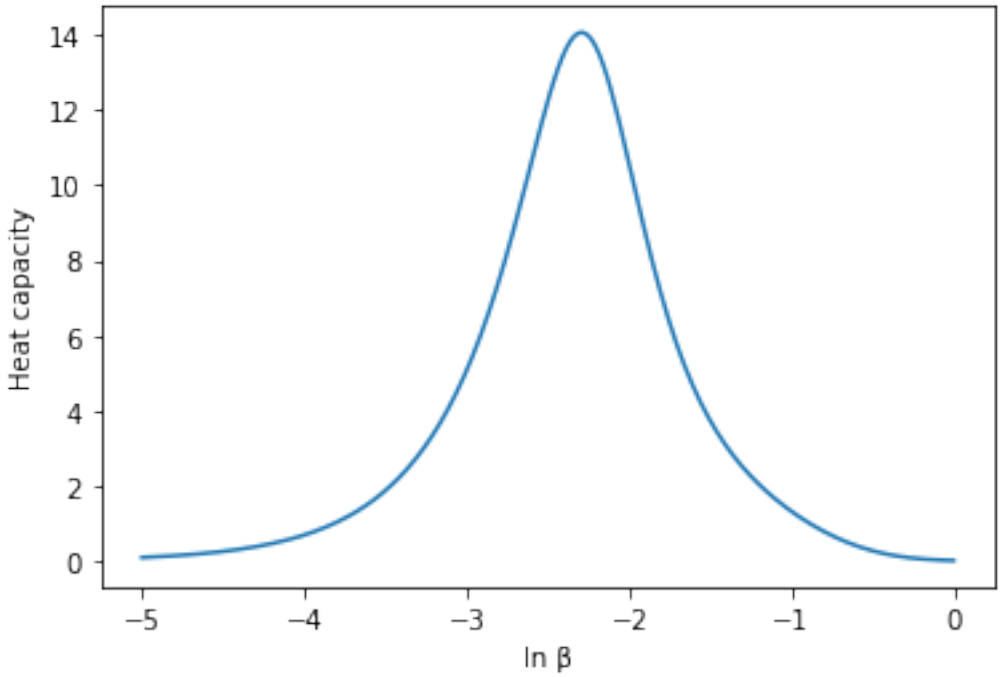
```
Z = lambda β: np.sum(gs * np.exp(-β * nEs))
```

Ensemble averages

```
βs = [np.exp(k) for k in np.linspace(-5, 0, 200)]
Eμ = lambda β: np.sum(nEs * gs * np.exp(-β * nEs)) / Z(β)
E2 = lambda β: np.sum(nEs**2 * gs * np.exp(-β * nEs)) / Z(β)
CV = lambda β: (E2(β) - Eμ(β)**2) * β**2
F  = lambda β: -np.log(Z(β)) / β
Sc = lambda β: β*Eμ(β) + np.log(Z(β))
```

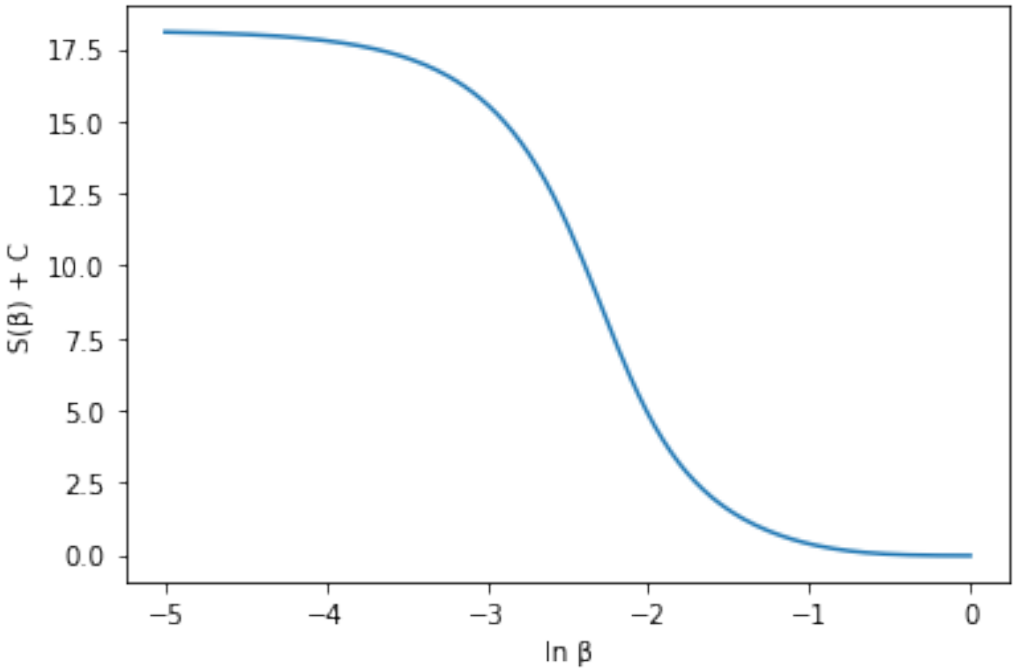
Heat capacity

```
plt.plot(np.log(βs), [CV(β) for β in βs])
plt.xlabel("ln β")
plt.ylabel("Heat capacity")
plt.show()
```



Entropy

```
plt.plot(np.log(βs), [Sc(β) for β in βs])
plt.xlabel("ln β")
plt.ylabel("S(β) + C")
plt.show()
```


17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
S(β) + C
−5
−4
−3
−2
−1
0
ln β