The Markov chain Monte Carlo (McMC) simulation of a photon gas

Contributions

- All
  - Identifying calculations to be run
  - Code reviews
- Gabe DeLong
  - Sketch of the algorithm
  - Suggesting some plotting details
- Enez Suyabatmaz
  - Writing and testing the pilot code
  - Suggesting the physical intepretations of question c
- Brian Zhao
  - Code clean-up and addition of plotting code
  - Suggesting the use of a varying num_step for different temperatures.

In [40]:

```
import numpy as np
import matplotlib.pyplot as plt

class PhotonGas:
    """
    A class for a simple single-mode non-interacting photon gas, where the only variable is the photon number N.
    The energy is given by E=N * epsilon, where epsilon is the energy of the photon mode.
    The number is allowed to freely vary in a grand canonical McMC simulation, and the Bose-Einstein distribution
    <N>(beta)=1/(exp(beta*epsilon)-1)
    should be reached at equilibrium,.
    """
    def __init__(self, be, n_init, mc_steps, eqm_steps=None):
        self.rng = np.random.default_rng() # Random number generator
        self.be = be # The only free variable is the beta*epsilon, used in the conditional acceptance
        self.configs = {} # {# of photons:# of times generated}, only accumulated after equilibration
        self.n_old = n_init
        self.n_new = None
        self.n_steps = mc_steps
        self.istep = 0

        # Throw away 80% of the trajectory as equilibration unless otherwise instructed.
        if (eqm_steps is None):
            self.eqm_steps = int(mc_steps*0.8)
        else:
            self.eqm_steps = int(eqm_steps)
        self.equilibrated = False

        self.U = self.get_U(self.n_old)
        self.U_traj = np.zeros(mc_steps)
        self.U_traj[self.istep] = self.U
        self.n_traj = np.zeros(mc_steps)
        self.n_traj[self.istep] = self.n_old

    def get_U(self, n):
        return self.be*n

    def decide_accept(self):
        u_new = self.get_U(self.n_new)
        u_old = self.get_U(self.n_old)
        if u_new <= u_old:
            # accept
            self.do_accept(u_new)
        else:
            # conditional acceptance if going up in energy
            rand = self.rng.uniform()
            if (np.exp(-(u_new-u_old)) > rand):
                self.do_accept(u_new)
            else:
                # If move rejected, accumulate the old configuration again to satisfy detailed balance
                if (self.equilibrated):
                    if self.n_old in self.configs.keys():
                        self.configs[self.n_old] += 1
                    else:
                        self.configs[self.n_old] = 1
                self.U = u_old

    def do_accept(self, u_new):
        self.n_old = self.n_new
```

```
        self.U = u_new
        if (self.equilibrated):
            if self.n_new in self.configs.keys():
                self.configs[self.n_new] += 1
            else:
                self.configs[self.n_new] = 1

    def do_mc_steps(self):
        while self.istep < self.n_steps-1:
            self.istep += 1
            if (self.istep >= self.eqm_steps):
                self.equilibrated = True
            self.generate_trial_move()
            self.decide_accept()
            self.U_traj[self.istep] = self.U
            self.n_traj[self.istep] = self.n_old

        # This can be either obtained from self.configs or equivalently from the trajectory over N,
        # the latter is more vectorized so we use that here
        self.mc_avg = np.average(self.n_traj[self.eqm_steps:])
        self.theo_avg = 1.0/(np.exp(self.be)-1) # The Bose-Einstein expectation

    def generate_trial_move(self):
        """
        Randomly increase or decrease the number of photons by 1
        """
        rand = self.rng.uniform()

        if (rand >= 0.5):
            self.n_new = self.n_old + 1
        else:
            self.n_new = self.n_old - 1

        # Photon number cannot be negative
        if self.n_new < 0:
            self.n_new = 0

    def plot_mc_traj(self):
        f,ax = plt.subplots(figsize=(12,6))
        ax.plot(self.n_traj[:])
        ax.axhline(self.mc_avg, linestyle='--', c='tab:orange', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}={mc_avg:
        ax.axhline(self.theo_avg ,linestyle='--', c='tab:green', label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}={theo
        ax.axvline(self.eqm_steps, linestyle='--', linewidth=1, c='k',label='Equilibration point')

        # Set up inset axis
        axins = ax.inset_axes([0.3,0.7,0.6,0.25])
        axins.plot(self.n_traj)
        axins.set_xlim([self.eqm_steps,self.n_steps])
        axins.axhline(self.mc_avg, linestyle='--', c='tab:orange', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}={mc_av
        axins.axhline(self.theo_avg ,linestyle='--', c='tab:green', label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}={t
        axins.set_ylim([self.mc_avg*0.8, self.mc_avg*1.2])

        ax.legend()
        ax.set_xlabel('MC iteration')
        ax.set_ylabel(r'$N_{{\mathrm{{phot.}}}}$')
        ax.indicate_inset_zoom(axins, edgecolor="black")
        _=ax.set_title(r'MC trajectory of $N_{{\mathrm{{phot.}}}}$ for a photon gas at $\beta\epsilon={i}$'.format(i=s

        return f,ax

class PhotonGas_alt1(PhotonGas):
    """
    Modified McMC algorithm where the old configuration is not re-counted if a new configuration is rejected.
    This gives qualitatively wrong behaviour at all temperatures, but it is more pronounced at low temperatures.
    """
    def decide_accept(self):
        u_new = self.get_U(self.n_new)
        u_old = self.get_U(self.n_old)
        if u_new <= u_old:
            # accept
            self.do_accept(u_new)
        else:
            # conditional acceptance
            rand = self.rng.uniform()
            if (np.exp(-(u_new-u_old)) > rand):
                self.do_accept(u_new)
            else:
                self.U = u_old

    def do_mc_steps(self):
        while self.istep < self.n_steps-1:
            self.istep += 1
            if (self.istep >= self.eqm_steps):
                self.equilibrated = True
            self.generate_trial_move()
```

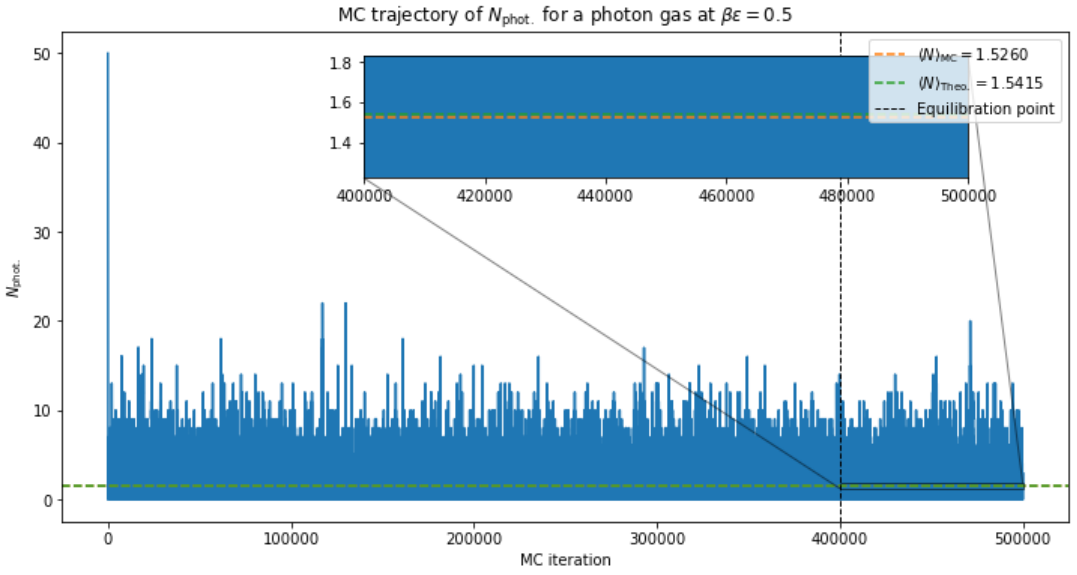
```
            self.decide_accept()
            self.U_traj[self.istep] = self.U
            self.n_traj[self.istep] = self.n_old

        self.mc_avg = 0.0
        for n in self.configs.keys():
            self.mc_avg += n*self.configs[n]
        self.mc_avg /= float(self.n_steps-self.eqm_steps)

        self.theo_avg = 1.0/(np.exp(self.be)-1)

class PhotonGas_alt2(PhotonGas):
    """
    Modified McMC algorithm where any move is accepted with 0.5 probability.
    This obviously does not obey detailed balance so convergence is never reached.
    """
    def decide_accept(self):
        u_new = self.get_U(self.n_new)
        u_old = self.get_U(self.n_old)

        rand = self.rng.uniform()
        if rand<=0.5:
            # accept
            self.do_accept(u_new)
        else:
            # reject
            if (self.equilibrated):
                if self.n_old in self.configs.keys():
                    self.configs[self.n_old] += 1
                else:
                    self.configs[self.n_old] = 1
```

In [35]:

```
a = PhotonGas(be=0.5, n_init=50, mc_steps=500000, eqm_steps=None)
a.do_mc_steps()
a.plot_mc_traj()
```

Out[35]:

```
(<Figure size 864x432 with 1 Axes>,
 <AxesSubplot:title={'center':'MC trajectory of $N_{\\mathrm{phot.}}$ for a photon gas at $\\beta\\epsilon=0.5$'}, xla
bel='MC iteration', ylabel='$N_{{\\mathrm{{phot.}}}}$'>)
```



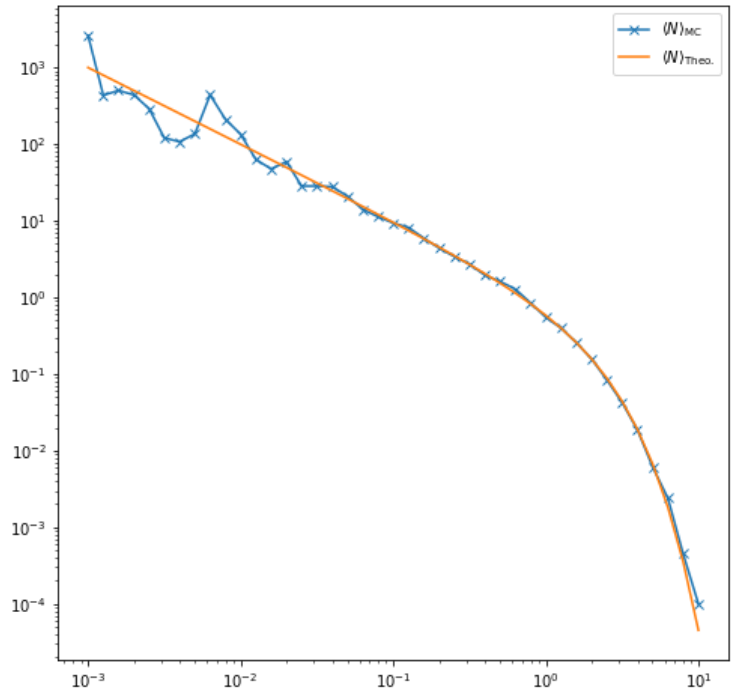
In [25]:

```
be_points = np.arange(-3,1.1,0.1)
be_points = 10**be_points
n_avg = np.zeros_like(be_points)
n_avg_theo = np.zeros_like(be_points)
# This uses the most steps for the challenging, high-temperature systems,
# while keeping the cost down for the easier, low-temperature systems
nsteps = 200000 + 2500000*np.exp(-10*be_points)
neqm = nsteps*0.8
for i,be in enumerate(be_points):
    a = PhotonGas(be=be, n_init=50, mc_steps=int(nsteps[i]), eqm_steps=int(neqm[i]))
    a.do_mc_steps()
    n_avg[i] = a.mc_avg
    n_avg_theo[i] = a.theo_avg
```

In [26]:

```
f,ax = plt.subplots(figsize=(8,8))
ax.loglog(be_points, n_avg, marker='x', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}$')
ax.loglog(be_points, n_avg_theo, label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}$')
ax.legend()
```

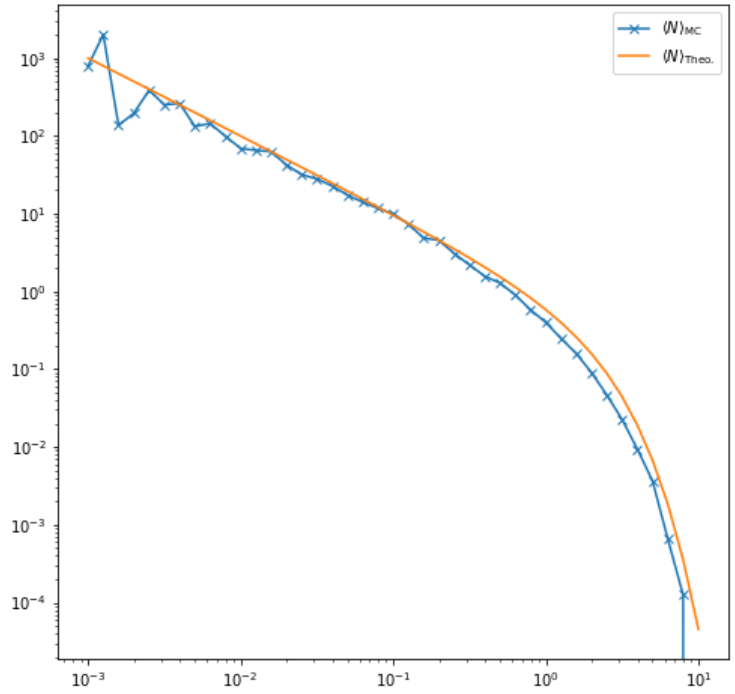
Out[26]: <matplotlib.legend.Legend at 0x7f090731c7c0>



In [27]:
```
be_points = np.arange(-3,1.1,0.1)
be_points = 10**be_points
n_avg = np.zeros_like(be_points)
n_avg_theo = np.zeros_like(be_points)
nsteps = 200000 + 2500000*np.exp(-10*be_points)
neqm = nsteps*0.8
for i,be in enumerate(be_points):
    a = PhotonGas_alt1(be=be, n_init=50, mc_steps=int(nsteps[i]), eqm_steps=int(neqm[i]))
    a.do_mc_steps()
    n_avg[i] = a.mc_avg
    n_avg_theo[i] = a.theo_avg
```

In [28]:
```
f,ax = plt.subplots(figsize=(8,8))
ax.loglog(be_points, n_avg, marker='x', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}$')
ax.loglog(be_points, n_avg_theo, label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}$')
ax.legend()
```

Out[28]: <matplotlib.legend.Legend at 0x7f09079e4dc0>

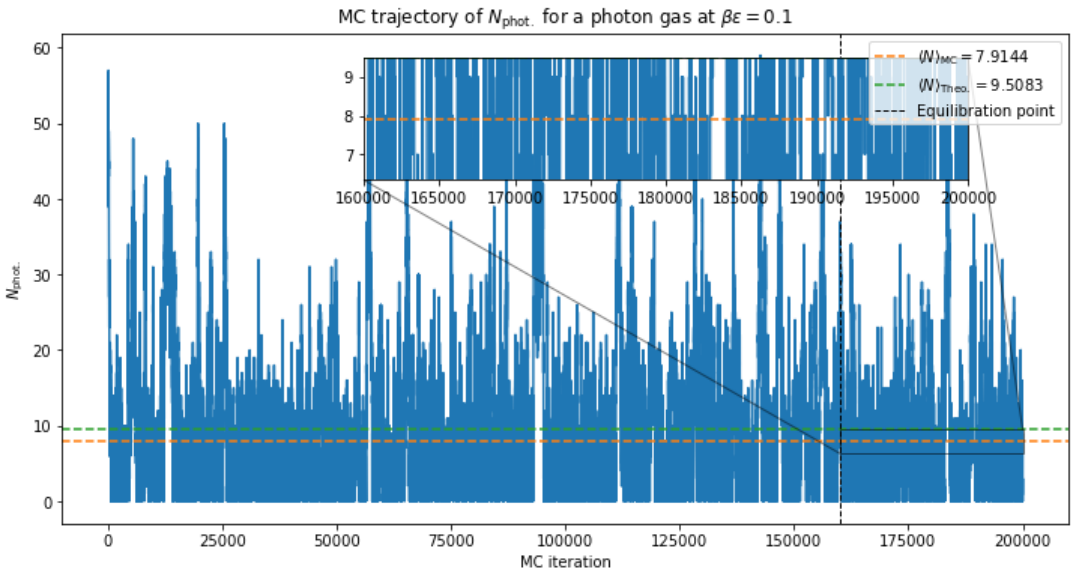


We can see that if repeated configurations were not accumulated as separate, the McMC averages are generally below the theoretical value, and the agreement becomes worse as temperature becomes lower. This is expected as the probability of rejecting an increase-

move is higher at lower temperatures. By not re-counting, this artificially weighs the accepted configurations, which are predominantly lower energy when the temperature is low, which explains why the averages are lower.

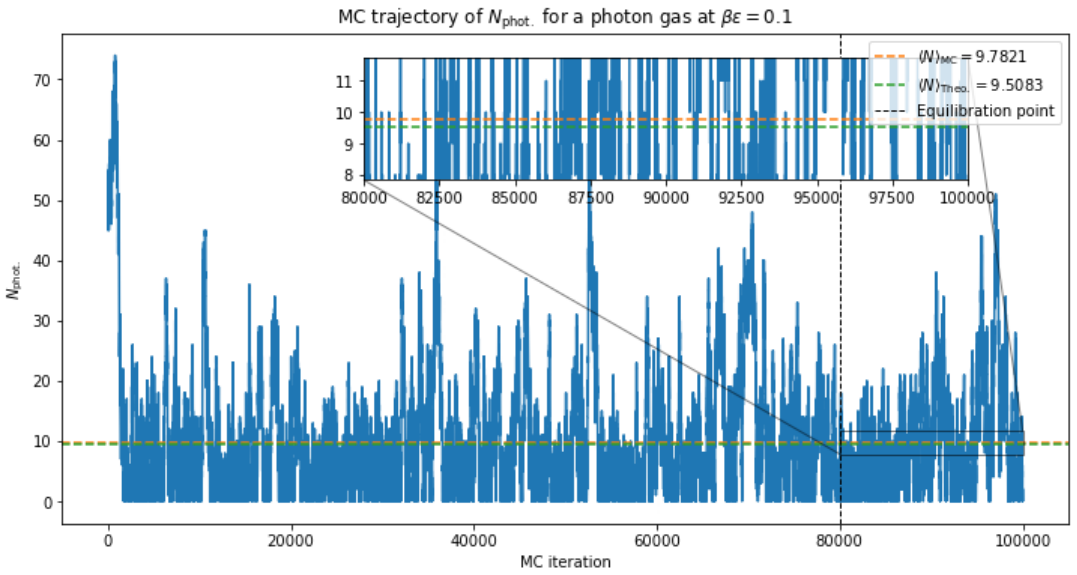
In [105…]

```
_, ax = a.plot_mc_traj()
```



In [99]:

```
f,ax = plt.subplots(figsize=(12,6))
ax.plot(a.n_traj[:])
ax.axhline(a.mc_avg, linestyle='--', c='tab:orange', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}={mc_avg:.4f}$'.forma
ax.axhline(a.theo_avg ,linestyle='--', c='tab:green', label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}={theo_avg:.4f}$
ax.axvline(a.eqm_steps, linestyle='--', linewidth=1, c='k',label='Equilibration point')
axins = ax.inset_axes([0.3,0.7,0.6,0.25])
axins.plot(a.n_traj[:])
axins.set_xlim([a.eqm_steps,a.n_steps])
axins.axhline(a.mc_avg, linestyle='--', c='tab:orange', label=r'$\langle N \rangle_{{\mathrm{{MC}}}}={mc_avg:.4f}$'.fo
axins.axhline(a.theo_avg ,linestyle='--', c='tab:green', label=r'$\langle N \rangle_{{\mathrm{{Theo.}}}}={theo_avg:.4
axins.set_ylim([a.mc_avg*0.8, a.mc_avg*1.2])
#xticklabels = np.arange(a.eqm_steps,a.n_steps,int((a.n_steps-a.eqm_steps)/len(ax.get_xticks()))-1)
#ax.set_xticklabels(xticklabels)
ax.legend()
ax.set_xlabel('MC iteration')
ax.set_ylabel(r'$N_{{\mathrm{{phot.}}}}$')
ax.indicate_inset_zoom(axins, edgecolor="black")
_=ax.set_title(r'MC trajectory of $N_{{\mathrm{{phot.}}}}$ for a photon gas at $\beta\epsilon={i}$'.format(i=a.be))
```

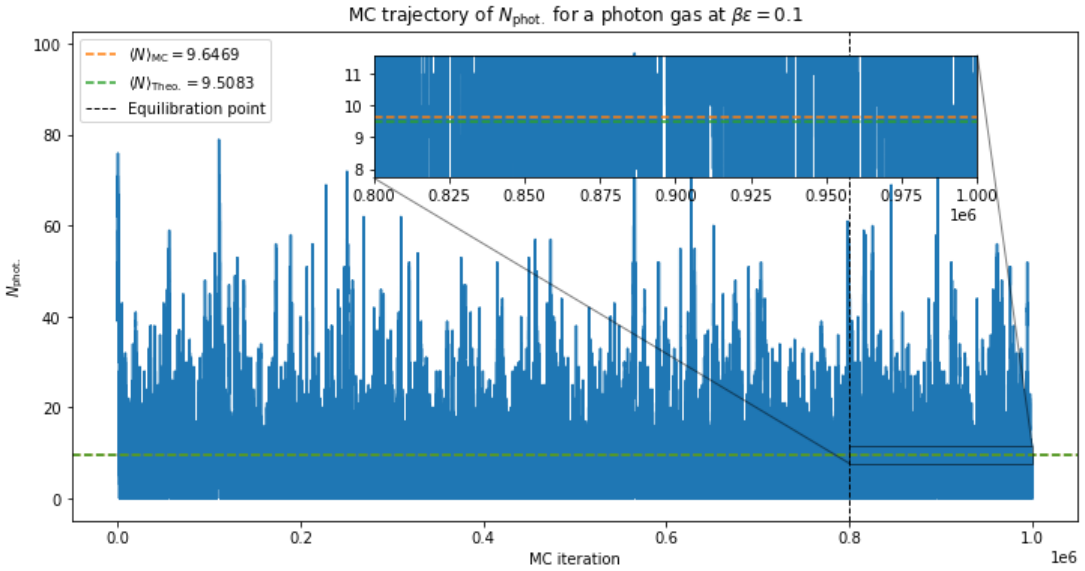


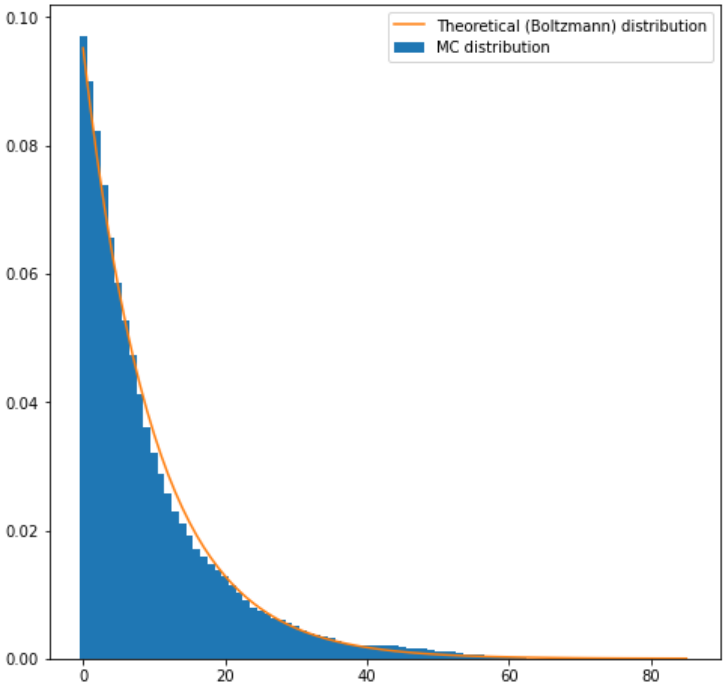
In [39]:

```
a = PhotonGas(be=0.1, n_init=50, mc_steps=1000000, eqm_steps=None)
a.do_mc_steps()
a.plot_mc_traj()

f,ax = plt.subplots(figsize=(8,8))
configs = np.array(sorted(a.configs.keys()))
prob = np.array([a.configs[_] for _ in configs],dtype='float64')
prob /= float(a.n_steps-a.eqm_steps)
```

```
theo_points = np.arange(0,configs[-1]+1)
theo_dist = np.array([np.exp(-a.be*_) for _ in theo_points]) * (1-np.exp(-a.be))

ax.bar(configs,prob,width=1,label='MC distribution')
ax.plot(theo_points,theo_dist,c='tab:orange',label='Theoretical (Boltzmann) distribution')
ax.legend()
```

Out[39]: <matplotlib.legend.Legend at 0x7f08f82a6fa0>





If the acceptance criterion were changed to randomly accept *any* move with 50% probability, detailed balance would be violated and convergence will never be reached, as the transition matrix would not be set up correctly to give the stationary state.

To see how detailed balance is violated, detailed balance demands that

$$p(\mathrm{o})\pi(\mathrm{o} \to \mathrm{n}) = p(\mathrm{n})\pi(\mathrm{n} \to \mathrm{o}),$$

where $\pi(\mathrm{o} \to \mathrm{n})$ is the transition probability from the old to the new configuration, and is decomposed into

$$\pi(\mathrm{o} \to \mathrm{n}) = \alpha(\mathrm{o} \to \mathrm{n})\mathrm{acc}(\mathrm{o} \to \mathrm{n}),$$

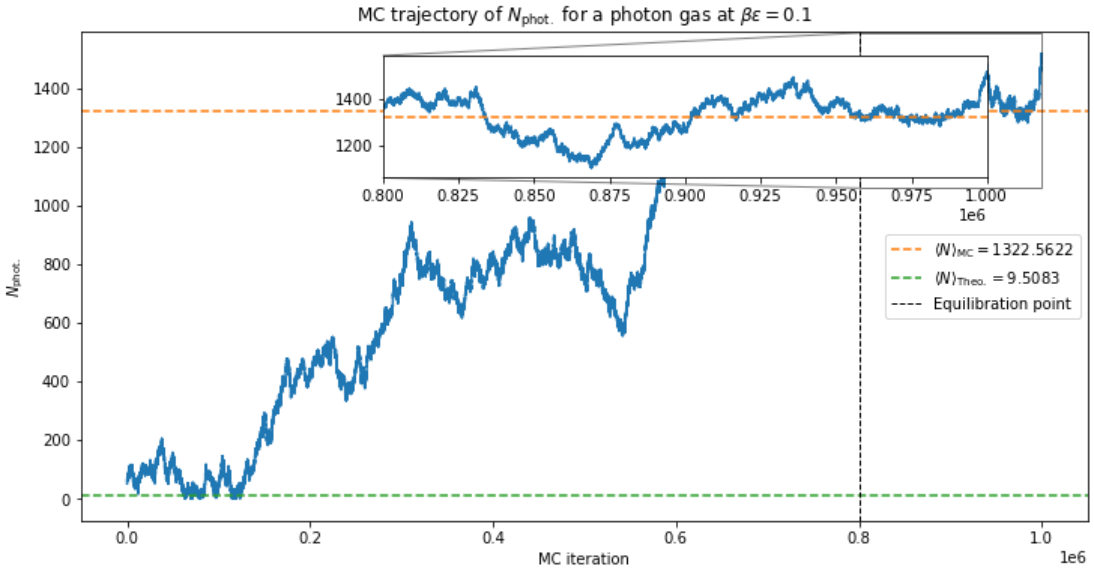
where $\alpha(\mathrm{o} \to \mathrm{n})$ is the probability of generating the transition, and $\mathrm{acc}(\mathrm{o} \to \mathrm{n})$ is the probability of accepting the transition. Assuming a symmetrical α, detailed balance therefore requires that

$$\frac{\mathrm{acc}(\mathrm{o} \to \mathrm{n})}{\mathrm{acc}(\mathrm{n} \to \mathrm{o})} = e^{-\beta[U(\mathrm{n}-U(\mathrm{o}))]},$$

however, by setting $\mathrm{acc} = 0.5$ throughout, the ratio would be always 1, violating conditions of detailed balance.

In [41]:
```
a = PhotonGas_alt2(be=0.1, n_init=50, mc_steps=1000000, eqm_steps=None)
a.do_mc_steps()
a.plot_mc_traj()
```

Out[41]:
```
(<Figure size 864x432 with 1 Axes>,
 <AxesSubplot:title={'center':'MC trajectory of $N_{\\mathrm{phot.}}$ for a photon gas at $\\beta\\epsilon=0.1$'}, xla
bel='MC iteration', ylabel='$N_{{\\mathrm{{phot.}}}}$'>)
```



In [ ]:
```
```