Chapter 13: Hybrid Monte Carlo

In this chapter we discuss the Hybrid Monte Carlo (HMC) algorithm that can efficiently generate gauge configurations with dynamical fermions. We go step-by-step and first introduce needed concepts.

Molecular Dynamics

The Hamiltonian $H(q,p)$ with canonical coordinate q and canonical momentum p is constant under time-evolution of p and q following the equations of motion

$$\dot{q} = \frac{\partial H}{\partial p}\,, \qquad \dot{p} = -\frac{\partial H}{\partial q}\,. \tag{1}$$

They can therefore be used to simulate the evolution of a microcanonical ensemble (where the energy is constant and specified).

Now consider a simple Hamiltonian

$$H(p,q) = \frac{1}{2}p^2 + V(q)\,, \tag{2}$$

then the equations of motions are

$$\dot{q} = p\,, \qquad \dot{p} = -V'(q)\,. \tag{3}$$

A simple numerical scheme to perform the Molecular Dynamics (MD) integration is the **Euler integrator**

$$p(t+dt) = p(t) - V'(q(t))dt\,, \tag{4}$$
$$q(t+dt) = q(t) + p(t)dt\,. \tag{5}$$

A better scheme is the **Leapfrog integrator**

$$p(t+dt/2) = p(t) - V'(q(t))dt/2\,, \tag{6}$$
$$q(t+dt) = q(t) + p(t+dt/2)dt\,, \tag{7}$$
$$p(t+dt) = p(t+dt/2) - V'(q(t+dt))dt/2 \tag{8}$$

which is symplectic, i.e., it preserves the $dp \wedge dq$. It preserves the energy to high accuracy and is reversible as you will show in your homework.

Homework: Show for the Leapfrog that

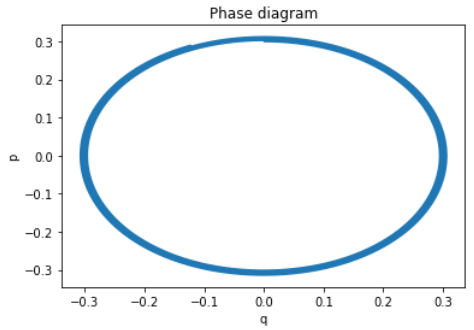
$$H(p(t+dt),q(t+dt)) - H(p(t),q(t)) = O(dt^3) \tag{9}$$

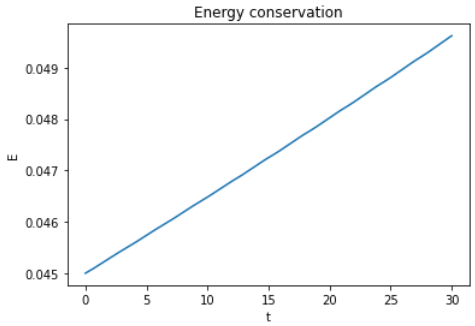
Show also that the Leapfrog is reversible, i.e., that evolving with $-dt$ after evolving with dt returns to the same identical (p,q).

Let us demonstrate these integrators in a simple example.

In [14]:
```
import gpt as g
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate

def V(q):
    return 1.0/2.0*q**2 + 1.0/4.0 * q**4

def dV(q):
    return q + q**3

def integrator_euler(p, q, dt):
    return p - dV(q)*dt, q + p*dt

def plot(p, q, eps, integrator):
    x = []
    y = []
    t = []
    E = []
    for i in range(10000):
        p, q = integrator(p, q, eps)
        x.append(q)
        y.append(p)
        t.append(i*eps)
        E.append(p**2./2.0 + V(q))

    fig, ax = plt.subplots()
    plt.title("Phase diagram")
    plt.xlabel("q")
    plt.ylabel("p")
    ax.plot(x,y)
    plt.show()

    fig, ax = plt.subplots()
    plt.title("Energy conservation")
    ax.ticklabel_format(style="sci",useOffset=False)
    plt.xlabel("t")
    plt.ylabel("E")
    ax.plot(t,E)
    plt.show()

    # check reversibility
```

```
    p, q = integrator(p, q, -eps)
    delta = ((p - y[-2])**2. + (q - x[-2])**2.)**0.5
    print("Reversibility test:",delta)

plot(0.3, 0, 0.003, integrator_euler)
```

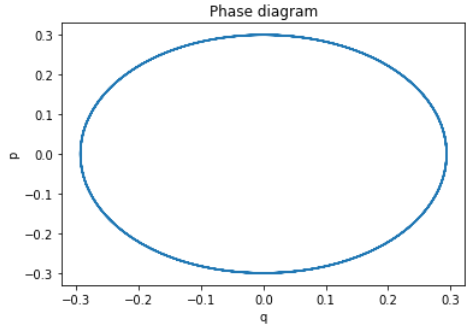


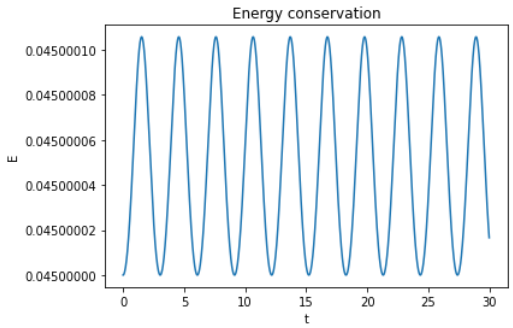



```
Reversibility test: 2.951175396656005e-06
```

In [99]:

```
def integrator_leapfrog(p, q, dt):
    p_half = p - dV(q)*dt/2.
    q_one = q + p_half*dt
    p_one = p_half - dV(q_one)*dt/2.
    return p_one, q_one

plot(0.3, 0, 0.003, integrator_leapfrog)
```






```
Reversibility test: 0.0
```

The path integral using Molecular Dynamics

Let us define a simple bosonic path integral

$$\langle O(q) \rangle = \frac{\int dq O(q) e^{-V(q)}}{\int dq e^{-V(q)}} = \frac{\int dq dp O(q) e^{-V(q)-p^2/2}}{\int dq dp e^{-V(q)-p^2/2}} \tag{10}$$

$$= \frac{\int dq dp O(q) e^{-H(p,q)}}{\int dq dp e^{-H(p,q)}} \tag{11}$$

If we now randomly pick momenta p following the distribution

$$e^{-p^2/2}$$

and use an exact MD integrator to integrate a trajectory, we do not change $H(p,q)$. If such a process were to be ergodic, i.e., any field configuration can be reached and reversible, then we can use the MD integration as a proposal step in the Metropolis algorithm of chapter 2. Since the energy is conserved, all proposed configurations should be accepted. Let us demonstrate this algorithm.

In [143…

```
rng = g.random("hmc")

ref_val = (
    scipy.integrate.quad(lambda q:q**2.*np.exp(-q**2./2.-q**4./4.),-np.infty,np.infty)[0]
    / scipy.integrate.quad(lambda q:np.exp(-q**2./2.-q**4./4.),-np.infty,np.infty)[0]
)

def md_integrate(rng, f, eps, trajectory_length, trajectories, integrator, verbose):
    q = 0.0
    ftot = 0.0
    dE2tot = 0.0
    for i in range(trajectories):
        p = rng.normal().real
        E0 = p**2./2. + V(q)
        for j in range(trajectory_length):
            p, q = integrator(p, q, eps)
        E1 = p**2./2. + V(q)
        dE2tot += (E1-E0)**2.
        ftot += f(q)
    if verbose:
        print("Energy changed by",(dE2tot / trajectories)**0.5)
    return ftot / trajectories

measurements = [md_integrate(rng, lambda q: q**2., 0.01, 100, 1000, integrator_leapfrog, j==0) for j in range(40)]
hmc_mean = np.mean(measurements)
hmc_err = np.std(measurements) / len(measurements)**0.5
print(hmc_mean,"+-",hmc_err)

print(ref_val)
```

```
GPT :   12129.165851 s : Initializing gpt.random(hmc,vectorized_ranlux24_389_64) took 0.000347853 s
Energy changed by 4.9812149431683414e-05
0.4693255931366978 +- 0.0026291580867840044
0.4679199169736833
```

The Hybrid Monte Carlo algorithm

We cannot make the step size in the above algorithm much larger since we have a systematic error due to the non-conservation of energy. This is, however, readily fixed by adding an accept-reject step. Let us demonstrate:

In [144…

```
def hmc_integrate(rng, f, eps, trajectory_length, trajectories, integrator, verbose):
    q = 0.0
    ftot = 0.0
    dE2tot = 0.0
    reject = 0
    accept = 0
    for i in range(trajectories):
        p = rng.normal().real
        E0 = p**2./2. + V(q)
        q0 = q
        for j in range(trajectory_length):
            p, q = integrator(p, q, eps)
        E1 = p**2./2. + V(q)
        if np.exp(E0-E1) < rng.uniform_real(min=0,max=1):
            reject += 1
            q = q0
        else:
            accept += 1
            dE2tot += (E1-E0)**2.
        ftot += f(q)
    if verbose:
        print("Energy changed by",(dE2tot / trajectories)**0.5)
        print("Acceptance rate",accept/(accept+reject))
    return ftot / trajectories

# can run with much larger stepsize, i.e., fewer steps within a trajectory, roughly same cost as MD above
# smaller statistical error compared to pure MD due to more uncorrelated configurations at same cost
measurements = [hmc_integrate(rng, lambda q: q**2., 0.8, 2, 50000, integrator_leapfrog, j==0) for j in range(40)]
hmc_mean = np.mean(measurements)
hmc_err = np.std(measurements) / len(measurements)**0.5
print(hmc_mean,"+-",hmc_err)

print(ref_val)
```

```
Energy changed by 0.371463895432758
Acceptance rate 0.80436
0.4676423573794918 +- 0.0009053735027992211
0.4679199169736833
```

Homework: Use the HMC algorithm to compute $\int dx x^4 e^{-x^4} / \int dx e^{-x^4}$.

Strength of the HMC

We can generate configurations that are to a large degree uncorrelated by evolving for a long time within a microcanonical ensemble. This performs an update that globally reduces correlation. At the same time, we approximately conserve the Hamiltonian, so that we have a good acceptance rate even for large global field configuration changes. This effectively reduces the autocorrelation compared to other global update schemes. Local updating schemes such as the Heatbath are also very effective but require a

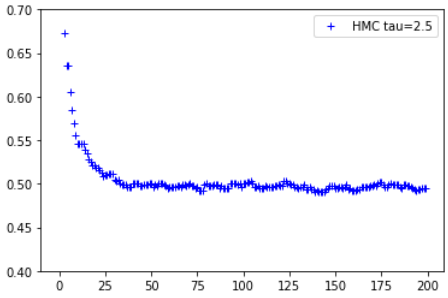
specific restricted local form of the action. The introduction of a fermion determinant, e.g., violates the local form of the action. The HMC on the other hand can perform global updates while only requiring knowledge of the derivative of the action with respect to the field configuration.

HMC generation of quenched ensemble

In [13]:

```
L = [8, 8, 8, 8]
grid = g.grid(L, g.double)

rng = g.random("test", "vectorized_ranlux24_24_64")
U = g.qcd.gauge.random(grid, rng)
Nd = len(U)

# conjugate momenta
mom = g.group.cartesian(U)

a0 = g.qcd.scalar.action.mass_term()
a1 = g.qcd.gauge.action.wilson(5.5)

def hamiltonian():
    a1v = a1(U)
    return a0(mom) + a1v, a1v

sympl = g.algorithms.integrator.symplectic

ip = sympl.update_p(mom, lambda: a1.gradient(U, U))
iq = sympl.update_q(U, lambda: a0.gradient(mom, mom))

mdint = sympl.OMF4(5, ip, iq)

metro = g.algorithms.markov.metropolis(rng)

def hmc(tau, mom):
    rng.normal_element(mom)
    accrej = metro(U)
    h0, s0 = hamiltonian()
    mdint(tau)
    h1, s1 = hamiltonian()
    return [accrej(h1, h0), s1 - s0, h1 - h0]

plaquette_hmc = []
accept = 0
total = 0
for it in range(200):
    plaq = g.qcd.gauge.plaquette(U)
    plaquette_hmc.append(plaq)
    a, dS, dH = hmc(1.5 if it < 10 else 2.5, mom)
    accept += a
    total += 1
    if it % 25 == 0:
        g.message(f"HMC {it} has P = {plaq}, dS = {dS}, acceptance = {accept/total}")

fig, ax = plt.subplots()

plt.ylim([0.4,0.7])
ax.plot(range(len(plaquette_hmc)), plaquette_hmc, marker='+', ls='', c='blue', label="HMC tau=2.5")

plt.legend()
plt.show()
```

```
GPT :    2335.016326 s : Initializing gpt.random(test,vectorized_ranlux24_24_64) took 4.60148e-05 s
GPT :    2336.024092 s : HMC 0 has P = 0.7984117298395389, dS = 10379.119118396331, acceptance = 1.0
GPT :    2357.513546 s : HMC 25 has P = 0.5101957086189 31, dS = 532.1766394544538, acceptance = 0.6923076923076923
GPT :    2384.033852 s : HMC 50 has P = 0.5002854277947527, dS = 624.5210875141056, acceptance = 0.7254901960784313
GPT :    2414.452550 s : HMC 75 has P = 0.49634571797119015, dS = 505.94901705106895, acceptance = 0.7763157894736842
GPT :    2447.146318 s : HMC 100 has P = 0.4998291900738143, dS = -188.5970112660725, acceptance = 0.7524752475247525
GPT :    2481.451864 s : HMC 125 has P = 0.5002291687006946, dS = 210.18674995772017, acceptance = 0.7619047619047619
GPT :    2509.303130 s : HMC 150 has P = 0.49520164333211647, dS = 10.240288826535107, acceptance = 0.7748344370860927
GPT :    2531.547358 s : HMC 175 has P = 0.5012029695511525, dS = 300.1770389723679, acceptance = 0.7727272727272727
```



HMC generation of dynamical ensemble

In order to generate a dynamical fermion ensemble, we would like to generate configurations in the path integral

$$Z = \int d[U] d\bar{\psi} d\psi e^{-S_G[U] - \sum_f \bar{\psi}_f D[U,m_f] \psi_f} = \int d[U] \left(\prod_f \det(D[U,m_f]) \right) e^{-S_G[U]}, \tag{12}$$

where the sum is over fermion flavors f with masses m_f. The determinant can then be re-expressed using

$$\det(M) = \int d\phi d\phi^\dagger e^{-\phi^\dagger M^{-1} \phi}\,, \tag{13}$$

which converges for positive definite M. A single Dirac matrix is not positive definite, however, due to γ_5-Hermiticity, one finds

$$\det(D) = \det(\gamma_5 D^\dagger \gamma_5) = \det(D^\dagger) \tag{14}$$

such that for two degenerate flavors, we find

$$\det(D[U,m])^2 = \det(D[U,m]D[U,m]^\dagger)\,. \tag{15}$$

Since the matrix $M = D[U,m]D[U,m]^\dagger$ is positive definite,

$$Z = \int d[U] \det(D[U,m])^2 e^{-S_G[U]} = \int d[U] d\phi d\phi^\dagger e^{-S_G[U] - \phi^\dagger (D[U,m]D[U,m]^\dagger)^{-1}\phi}\,. \tag{16}$$

The bosonic fields ϕ are called **pseudo-fermion** fields.

If one wants to introducing a single additional flavor (such as a strange quark), one may use rational approximations of the inverse square root as we have studied in the last chapter. For now, however, we focus on the important case of two degenerate quark flavors that in the real world, we identify with the up and down quarks with degenerate mass $m = m_u = m_d$. For high-precision calculations, we will then later add corrections in the small $m_u - m_d$.

The concrete algorithm for the two-flavor HMC proceeds as follows. We identify

$$\eta = D[U,m]^{-1}\phi \tag{17}$$

such that

$$Z = \int d[U] d\phi d\phi^\dagger e^{-S_G[U] - \eta^\dagger \eta}\,. \tag{18}$$

We then randomly draw a field configuration for η from the trivial distribution

$$e^{-\eta^\dagger \eta} \tag{19}$$

and compute

$$\phi = D[U,m]\eta\,. \tag{20}$$

We then introduce conjugate momenta for the gauge fields, draw them randomly from a distribution

$$e^{-p^2/2} \tag{21}$$

and perform the MD integration using the action

$$S[U,p] = S_G[U] + \phi^\dagger (D[U,m]D[U,m]^\dagger)^{-1}\phi + p^2/2\,. \tag{22}$$

We complete this process with an accept-reject step.

We show the implementation of an HMC for two dynamical degenerate quark flavors below:

In [2]:
```
import gpt as g
```

In [18]:
```
L = [8, 8, 8, 16]
grid = g.grid(L, g.double)

rng = g.random("test", "vectorized_ranlux24_24_64")
U = [g.project(x,"defect") for x in g.qcd.gauge.random(grid, rng)]
Nd = len(U)

# conjugate momenta
U_mom = g.group.cartesian(U)

rng.normal_element(U_mom)

# reproduce https://arxiv.org/pdf/hep-lat/0411006.pdf
a0 = g.qcd.scalar.action.mass_term()
a1 = g.qcd.gauge.action.improved_with_rectangle(0.8,-1.4069)
D_m = g.qcd.fermion.mobius(U, M5=1.8, mass=0.04, Ls=12, b=1., c=0.,
                           boundary_phases=[1,1,1,-1])
D_pv = g.qcd.fermion.mobius(U, M5=1.8, mass=1.0, Ls=12, b=1., c=0.,
                            boundary_phases=[1,1,1,-1])

inv = g.algorithms.inverter
pc = g.qcd.fermion.preconditioner
g.default.set_verbose("cg_convergence", False)
g.default.set_verbose("cg", False)
cg = inv.cg({"eps": 1e-8, "maxiter": 1000})

a2 = g.qcd.pseudofermion.action.two_flavor_ratio_evenodd_schur([D_m, D_pv], cg)

P = g.vspincolor(D_m.F_grid_eo)
fields = U + [P]

sympl = g.algorithms.integrator.symplectic

show_force = False

def total_force():
    global show_force
    gauge_force = a1.gradient(U,U)
    fermion_force = a2.gradient(fields,U)
```

```
    if show_force:
        g.message("gauge force", (sum(g.norm2(gauge_force)) / 4 / P.grid.gsites)**0.5)
        g.message("fermion force", (sum(g.norm2(fermion_force)) / 4 / P.grid.gsites)**0.5)
        show_force = False
    return [g(x+y) for x,y in zip(gauge_force,fermion_force)]

ipU = sympl.update_p(U_mom, lambda: total_force())
iqUM = sympl.update_q(U, lambda: a0.gradient(U_mom, U_mom))

mdint = sympl.OMF4(5, ipU, iqUM)

metro = g.algorithms.markov.metropolis(rng)

def hamiltonian(draw):
    if draw:
        rng.normal_element(U_mom)
        a0v = a0(U_mom)
        a1v = a1(U)
        a2v = a2.draw(fields, rng)
    else:
        a0v = a0(U_mom)
        a1v = a1(U)
        a2v = a2(fields)
    return a0v + a1v + a2v, a1v + a2v


def hmc(tau):
    accrej = metro(U)
    h0, s0 = hamiltonian(True)
    mdint(tau)
    h1, s1 = hamiltonian(False)
    return [accrej(h1, h0), s1 - s0, h1 - h0]


plaquette_hmc = []
accept = 0
total = 0
for it in range(50):
    plaq = g.qcd.gauge.plaquette(U)
    plaquette_hmc.append(plaq)
    show_force = it % 10 == 0
    a, dS, dH = hmc(1.3 if it < 10 else 1.5)
    accept += a
    total += 1
    if it % 1 == 0:
        g.message(f"HMC {it} has P = {plaq}, dS = {dS}, dH = {dH}, acceptance = {accept/total}")
```

```
GPT :    3226.733732 s : Initializing gpt.random(test,vectorized_ranlux24_24_64) took 3.60012e-05 s
GPT :    3235.798564 s : gauge force 1.7894410341122076
GPT :    3235.801223 s : fermion force 0.17477311206344134
GPT :    3321.598793 s : HMC 0 has P = 0.7977536349278497, dS = 14125.963963725837, dH = -0.31147062371019274, acceptance = 1.0
GPT :    3418.014995 s : HMC 1 has P = 0.7239331849940017, dS = 8454.722610203899, dH = 0.05969532590825111, acceptance = 1.0
GPT :    3533.763383 s : HMC 2 has P = 0.6943358474812369, dS = 5455.348705547745, dH = 0.0858488124795258, acceptance = 1.0
GPT :    3656.958261 s : HMC 3 has P = 0.6760186855667949, dS = 3259.6686018415494, dH = 0.06032693374436349, acceptance = 1.0
GPT :    3792.405839 s : HMC 4 has P = 0.6654805318112572, dS = 1969.0264017770533, dH = 0.05946384253911674, acceptance = 0.8
GPT :    3932.168001 s : HMC 5 has P = 0.6654805318112572, dS = 2299.6916684884345, dH = 0.03148992813657969, acceptance = 0.8333333333333334
GPT :    4077.065966 s : HMC 6 has P = 0.6574594684876843, dS = 704.2069775389973, dH = 0.05995966272894293, acceptance = 0.8571428571428571
GPT :    4228.510042 s : HMC 7 has P = 0.6545793673319614, dS = 930.5410016970709, dH = 0.3986480486346409, acceptance = 0.875
GPT :    4371.451943 s : HMC 8 has P = 0.6518778480084265, dS = 602.6918807112379, dH = 0.0027719002682715654, acceptance = 0.8888888888888888
GPT :    4515.562474 s : HMC 9 has P = 0.6503448896522969, dS = 407.39265598147176, dH = 0.009069292107596993, acceptance = 0.9
GPT :    4521.970773 s : gauge force 1.814392301308143
GPT :    4521.972303 s : fermion force 0.21589526829019431
GPT :    4658.327603 s : HMC 10 has P = 0.6489356239088632, dS = -6.288312049699016, dH = 0.2598933868575841, acceptance = 0.9090909090909091
GPT :    4805.961386 s : HMC 11 has P = 0.6494073794538987, dS = 164.83559356234036, dH = 0.5142212266800925, acceptance = 0.8333333333333334
GPT :    4966.673197 s : HMC 12 has P = 0.6494073794538987, dS = 114.54714150133077, dH = 0.0010217524832114577, acceptance = 0.8461538461538461
GPT :    5122.601279 s : HMC 13 has P = 0.6486879110260968, dS = 19.57796512846835, dH = -0.28001402865629643, acceptance = 0.8571428571428571
GPT :    5282.340239 s : HMC 14 has P = 0.6486526939960445, dS = 231.14872455596924, dH = 0.6017490805825219, acceptance = 0.8666666666666667
GPT :    5446.001001 s : HMC 15 has P = 0.6483896283156315, dS = -82.60883781616576, dH = -0.20881832262966782, acceptance = 0.875
GPT :    5600.534252 s : HMC 16 has P = 0.6485373676568366, dS = 361.97880395909306, dH = -0.06803504924755543, acceptance = 0.8823529411764706
GPT :    5763.193410 s : HMC 17 has P = 0.647554757587237, dS = -92.8522675470449, dH = -0.4093259364599362, acceptance = 0.8888888888888888
GPT :    5918.414479 s : HMC 18 has P = 0.6479524902950299, dS = -30.798952600336634, dH = -0.2993377334205434, acceptance = 0.8947368421052632
GPT :    6067.051604 s : HMC 19 has P = 0.6477996566889651, dS = 410.621951642097, dH = 0.16502909280825406, acceptance = 0.9
GPT :    6073.501520 s : gauge force 1.8237897700468328
GPT :    6073.502749 s : fermion force 0.21848204317890033
GPT :    6228.876946 s : HMC 20 has P = 0.6467007192206925, dS = -432.8139798906632, dH = -0.02418494736775756, acceptance = 0.9047619047619048
GPT :    6383.483695 s : HMC 21 has P = 0.6483591797810356, dS = -99.07440231973305, dH = 0.4043482915731147, acceptance = 0.9090909090909091
GPT :    6551.262866 s : HMC 22 has P = 0.6481523322089229, dS = 264.48225277871825, dH = -0.7522227831650525, acceptance = 0.9130434782608695
GPT :    6734.099637 s : HMC 23 has P = 0.6473319358700473, dS = -451.6132031325251, dH = -0.3410511414986104, acceptance = 0.9166666666666666
GPT :    6895.052412 s : HMC 24 has P = 0.6481366937251823, dS = 153.51149366027676, dH = -0.17425023810938, acceptance = 0.92
GPT :    7042.484981 s : HMC 25 has P = 0.6476120721631183, dS = 114.00059209542815, dH = 0.10573470173403621, acceptance = 0.9230769230769231
GPT :    7188.268112 s : HMC 26 has P = 0.6469252751582244, dS = -37.47324684332125, dH = 0.12481406063307077, acceptance = 0.9259259259259259
GPT :    7348.977756 s : HMC 27 has P = 0.6474102497423976, dS = 11.777054377365857, dH = -0.27457939402665943, acceptance = 0.9285714285714286
GPT :    7501.144125 s : HMC 28 has P = 0.6472965110551507, dS = 365.84789683669806, dH = 0.3019059435464442, acceptance = 0.896551724137931
GPT :    7641.082958 s : HMC 29 has P = 0.6472965110551507, dS = -148.8018067954108, dH = 0.3522139305714518, acceptance = 0.9
GPT :    7647.577584 s : gauge force 1.8179334205626891
GPT :    7647.579137 s : fermion force 0.21865133550525476
GPT :    7776.533566 s : HMC 30 has P = 0.6476223889322725, dS = -328.71844514412805, dH = 0.14074772945605218, acceptance = 0.9032258064516129
GPT :    7914.734084 s : HMC 31 has P = 0.6486266153624308, dS = 596.4052893508924, dH = 0.4665072920033708, acceptance = 0.90625
GPT :    8065.826109 s : HMC 32 has P = 0.6471191168076046, dS = 337.1160273979185, dH = -0.046541511313989776, acceptance = 0.9090909090909091
GPT :    8210.746242 s : HMC 33 has P = 0.6458873504584405, dS = 423.938943657442, dH = 0.1153583952691406, acceptance = 0.9117647058823529
GPT :    8355.740433 s : HMC 34 has P = 0.6453236142595575, dS = 300.71327484829817, dH = -0.10330085339955986, acceptance = 0.9142857142857143
GPT :    8516.198655 s : HMC 35 has P = 0.6440978572818944, dS = -407.1902210016269, dH = 0.19129672774579376, acceptance = 0.9166666666666666
GPT :    8693.174999 s : HMC 36 has P = 0.6449578954831817, dS = -31.844088866491802, dH = 0.009782907553301714, acceptance = 0.9189189189189189
GPT :    8857.137077 s : HMC 37 has P = 0.6453688570555499, dS = -449.8061942004133, dH = -0.6390336923068389, acceptance = 0.9210526315789473
GPT :    9017.438698 s : HMC 38 has P = 0.6476854574683141, dS = 145.2229589461349, dH = 0.1784240605775267, acceptance = 0.9230769230769231
GPT :    9176.537990 s : HMC 39 has P = 0.646992399179136, dS = 204.34257278998848, dH = 0.48368108842987567, acceptance = 0.925
GPT :    9183.389993 s : gauge force 1.8206442419484803
GPT :    9183.391684 s : fermion force 0.21848474071110654
GPT :    9343.047472 s : HMC 40 has P = 0.6464738396823121, dS = -64.2523499049712, dH = -0.2582530634244904, acceptance = 0.926829268292683
GPT :    9489.572929 s : HMC 41 has P = 0.6463921827666391, dS = -129.29299051046837, dH = -0.055380396428518, acceptance = 0.9285714285714286
```

```
GPT :    9619.297308 s : HMC 42 has P = 0.6473024528149066, dS = -427.43147500627674, dH = 0.2026908218394965, acceptance = 0.9069767441860465
GPT :    9755.924778 s : HMC 43 has P = 0.6473024528149066, dS = 56.265666058287024, dH = -0.15304966259282082, acceptance = 0.9090909090909091
GPT :    9885.403553 s : HMC 44 has P = 0.6469592309616699, dS = -82.33248552470468, dH = 0.10347239428665489, acceptance = 0.9111111111111111
GPT :   10014.447419 s : HMC 45 has P = 0.6469459876815528, dS = 42.72129428270273, dH = 0.09759782219771296, acceptance = 0.9130434782608695
GPT :   10151.242527 s : HMC 46 has P = 0.6468259260913222, dS = 218.82877777528483, dH = 0.3625646489672363, acceptance = 0.8936170212765957
GPT :   10288.244088 s : HMC 47 has P = 0.6468259260913222, dS = 254.30653610033914, dH = -0.22636164270807058, acceptance = 0.8958333333333334
GPT :   10432.206801 s : HMC 48 has P = 0.6459083262779279, dS = -96.70608586398885, dH = 0.05700209201313555, acceptance = 0.8979591836734694
GPT :   10590.198710 s : HMC 49 has P = 0.6463155079123528, dS = -506.70121625764295, dH = -0.42136163369286805, acceptance = 0.9
```

In [22]:

```
fig, ax = plt.subplots()

plt.ylim([0.6,0.8])
ax.plot(range(len(plaquette_hmc)), plaquette_hmc, marker='+', ls='', c='blue', label="HMC")
ax.hlines(0.646561,0,len(plaquette_hmc),label="expected")

plt.legend()
plt.show()
```