Chapter 4: scalar field theory on a lattice

Motivation

- So far we have considered one-particle Hamiltonians in one dimension. More interesting systems occur, e.g., when considering possible theories of the fundamental interactions of nature. Experimentally, we observe inelastic processes which change one set of incoming to a different set of outgoing particles. Example: radioactive decay, pion to muon decays in the atmosphere due to cosmic rays, ...
- We will show today that the Hilbert space of a quantum field theory is much better suited to describe such theories.
- In the next chapter, we will then more systematically explore the space of interesting field theories by starting from a thorough discussion of symmetries.

Hamiltonian of a real scalar quantum field theory in one spatial dimension

- Consider a real scalar quantum field theory with one spatial dimension with periodic boundary conditions on a spatial lattice with n_s positions defined by the Lagrangian

$$L = \sum_{x=0}^{n_s-1} \frac{1}{2}\phi(x,t)\left[\overleftarrow{\partial}_t\overrightarrow{\partial}_t + \Delta^2 - m^2\right]\phi(x,t) \tag{1}$$

with $x \in \{0,\dots,n_s-1\}$, $t, m \in \mathbb{R}$, and

$$\Delta^2\phi(x,t) \equiv \phi(x+1,t)+\phi(x-1,t)-2\phi(x,t) \tag{2}$$

and $\phi(-1,t)=\phi(n_s-1,t)$, $\phi(n_s,t)=\phi(0,t)$.

- The canonical momentum for the field is then given by the appropriate functional derivative

$$\pi(x,t) = \partial_{\partial_t\phi(x,t)}L = \partial_t\phi(x,t) \tag{3}$$

and the Euler-Lagrange equation

$$\partial_{\phi(x,t)}L = \partial_t\pi(x,t) \tag{4}$$

which is

$$\begin{aligned} 0 &= \partial_t^2\phi(x,t) - \partial_{\phi(x,t)}\sum_{x=0}^{n_s-1}\frac{1}{2}\phi(x,t)\left[\phi(x+1,t)+\phi(x-1,t)-(2+m^2)\phi(x,t)\right] && (5)\\ &= \partial_t^2\phi(x,t) - \left[\phi(x+1,t)+\phi(x-1,t)-(2+m^2)\phi(x,t)\right] && (6)\\ &= (\partial_t^2-\Delta^2+m^2)\phi(x,t)\,. && (7)\end{aligned}$$

- Next, we consider free-field solutions by first performing a discrete Fourier transform

$$\phi(x,t) = \frac{1}{\sqrt{2\pi n_s}}\int dE\sum_{k=0}^{n_s-1} e^{2\pi ixk/n_s}e^{itE}\tilde\phi(k,E) \tag{8}$$

yielding

$$\left(-E^2+\left(2\sin\left(\frac{\pi k}{n_s}\right)\right)^2+m^2\right)\tilde\phi(k,E)=0\,. \tag{9}$$

- Defining

$$w_k = \sqrt{m^2+\left(2\sin\left(\frac{\pi k}{n_s}\right)\right)^2} \tag{10}$$

we can then write the free field solution

$$f_k(x,t) \equiv \frac{1}{\sqrt{2w_kn_s}}e^{-iw_kt+i2\pi xk/n_s} \tag{11}$$

with

$$(\partial_t^2-\Delta^2+m^2)f_k(x,t)=0 \tag{12}$$

and

$$\begin{aligned}\sum_{x=0}^{n_s-1} f_k^*(x,t)f_{k'}(x,t) &= \frac{1}{2w_k}\delta_{k,k'}\,, && (13)\\ \sum_{x=0}^{n_s-1} f_k(x,t)f_{k'}(x,t) &= \frac{1}{2w_k}\delta_{k,-k'}e^{-2iw_kt}\,. && (14)\end{aligned}$$

- To quantize the theory, we first make the ansatz for the real field

$$\phi(x,t) = \sum_{k=0}^{n_s-1} \left[f_k(x,t)\hat{a}_k + f_k^*(x,t)\hat{a}_k^\dagger \right] \tag{15}$$

and operators $\hat{a}_k$ such that

$$\pi(x,t) = -i \sum_{k=0}^{n_s-1} w_k \left[f_k(x,t)\hat{a}_k - f_k^*(x,t)\hat{a}_k^\dagger \right] . \tag{16}$$

Then

$$\hat{a}_k = \sum_{x=0}^{n_s-1} f_k^*(x,t)(w_k\phi(x,t) + i\pi(x,t)) . \tag{17}$$

We quantize the fields canonically, i.e., require

$$[\phi(x,t), \pi(y,t)] = i\delta_{x,y} , \qquad [\phi(x,t), \phi(y,t)] = [\pi(x,t), \pi(y,t)] = 0 \tag{18}$$

yielding

$$[\hat{a}_k, \hat{a}_{k'}] = \sum_{x,y=0}^{n_s-1} f_k^*(x,t) f_{k'}^*(y,t) [w_k\phi(x,t) + i\pi(x,t), w_{k'}\phi(y,t) + i\pi(y,t)] \tag{19}$$

$$= \sum_{x,y=0}^{n_s-1} f_k^*(x,t) f_{k'}^*(y,t) i \left[w_{k'}[\pi(x,t), \phi(y,t)] + w_k[\phi(x,t), \pi(y,t)] \right] \tag{20}$$

$$= \sum_{x=0}^{n_s-1} f_k^*(x,t) f_{k'}^*(x,t) \left[w_{k'} - w_k \right] = 0 \tag{21}$$

and also

$$[\hat{a}_k, \hat{a}_{k'}^\dagger] = \sum_{x,y=0}^{n_s-1} f_k^*(x,t) f_{k'}(y,t) i \left[w_{k'}[\pi(x,t), \phi(y,t)] - w_k[\phi(x,t), \pi(y,t)] \right] \tag{22}$$

$$= \sum_{x=0}^{n_s-1} f_k^*(x,t) f_{k'}(x,t) \left[w_{k'} + w_k \right] = \delta_{k,k'} . \tag{23}$$

- Then Hamiltonian is given by a Legendre transformation of the Lagrangian

$$H = -L + \sum_{x=0}^{n_s-1} \pi(x,t) \partial_{\pi(x,t)} L \tag{24}$$

$$= \sum_{x=0}^{n_s-1} \frac{1}{2} \phi(x,t) \left[\overleftarrow{\partial}_t \overrightarrow{\partial}_t - \Delta^2 + m^2 \right] \phi(x,t)$$

$$= \sum_{x,k,k'=0}^{n_s-1} \frac{1}{2} \left[f_k(x,t)\hat{a}_k + f_k^*(x,t)\hat{a}_k^\dagger \right] \left[\overleftarrow{\partial}_t \overrightarrow{\partial}_t - \partial_t^2 \right] \left[f_{k'}(x,t)\hat{a}_{k'} + f_{k'}^*(x,t)\hat{a}_{k'}^\dagger \right] \tag{25}$$

$$= \sum_{x,k,k'=0}^{n_s-1} \frac{1}{2} \left[f_k(x,t)\hat{a}_k + f_k^*(x,t)\hat{a}_k^\dagger \right] w_{k'}^2 \left[f_{k'}(x,t)\hat{a}_{k'} + f_{k'}^*(x,t)\hat{a}_{k'}^\dagger \right]$$

$$- \sum_{x,k,k'=0}^{n_s-1} \frac{1}{2} \left[f_k(x,t)\hat{a}_k - f_k^*(x,t)\hat{a}_k^\dagger \right] w_{k'} w_{k'} \left[f_{k'}(x,t)\hat{a}_{k'} - f_{k'}^*(x,t)\hat{a}_{k'}^\dagger \right] \tag{26}$$

$$= \frac{1}{2} \sum_{k=0}^{n_s-1} w_k \left[\hat{a}_k \hat{a}_k^\dagger + \hat{a}_k^\dagger \hat{a}_k \right] \tag{27}$$

$$= \sum_{k=0}^{n_s-1} w_k \left[\frac{1}{2} + \hat{a}_k^\dagger \hat{a}_k \right] . \tag{28}$$

- This looks like a Harmonic oscillator for each mode number k. Note: we have a vacuum energy offset $E_0 = \frac{1}{2} \sum_{k=0}^{n_s-1} w_k$.
- Beyond the vacuum energy, there is a finite contribution of energy ω_k for each mode k multiplied by the occupation number operator

$$\hat{n}_k \equiv \hat{a}_k^\dagger \hat{a}_k . \tag{29}$$

- We therefore have found a theory which can describe an aribtrary number of particles in mode k that each contribute energy ω_k.
- We can write for small momenta

$$p_k = \frac{2\pi}{n_s} k$$

  the energies as

$$\omega_k^2 \approx m^2 + p_k^2$$

  which suggests that we found a theory of non-interacting particles that each obey the relativistic energy-momentum relation. In the next chapter, we illustrate the origin of this phenomenon.

Global and local Metropolis update for scalar fields

We will next discretize also time in n_t steps with periodic boundary conditions, as we did for the one-dimensional Hamiltonian examples, and study the Euclidean action

$$S_E(\phi) = \frac{1}{2}\sum_{x=0}^{n_s-1}\sum_{t=0}^{n_t-1}\phi(x,t)(-\Delta_t^2 - \Delta_x^2 + m^2)\phi(x,t) \tag{30}$$

where we used partial integration to identify $\overleftarrow{\partial_t}\,\overrightarrow{\partial_t} = -\partial_t^2$ and

$$\Delta_x^2\phi(x,t) = \phi(x+1,t) + \phi(x-1,t) - 2\phi(x,t)\,, \tag{31}$$
$$\Delta_t^2\phi(x,t) = \phi(x,t+1) + \phi(x,t+1) - 2\phi(x,t)\,. \tag{32}$$

We demonstrate how to generate a Markov chain with probability distribution $p(\phi) \propto e^{-S_E(\phi)}$ using both global and local updates. For this, we will use for the first time the high-performance library GPT (https://github.com/lehner/gpt).

In [22]:
```
import gpt as g
import numpy as np
from matplotlib import pyplot as plt
```

In [3]:
```
L = [32,64]
grid = g.grid(L, g.double)
grid_eo = g.grid(L, g.double, g.redblack)
```

In [4]:
```
field = g.real(grid)
field[:] = 0
```

In [5]:
```
rng = g.random("seed13")
```

```
GPT :       0.806299 s : Initializing gpt.random(seed13,vectorized_ranlux24_389_64) took 0.000421047 s
```

In [6]:
```
rng.element(field)
```

Out[6]:
```
lattice(ot_real_additive_group,double)
```

In [7]:
```
def S(phi, m0):
    action = 0.5*g.norm2(phi)*m0**2
    for mu in range(2):
        action -= 0.5 * g.inner_product(phi, g.cshift(phi,mu,1) + g.cshift(phi,mu,-1) - 2.*phi)
    return action

print(S(field, 0.1))
```

```
(327.52809672707235+0j)
```

In [8]:
```
def S_field(phi, m0):
    # gather in afld[x] everything that changes if phi[x] changes
    afld = g(0.5 * g.adj(phi)*phi*(m0**2 + 4.0))
    for mu in range(2):
        afld -= g.adj(phi) * (g.cshift(phi,mu,1) + g.cshift(phi,mu,-1))
    return afld

print(S(field,0.1))

# test change in single point
field[12,17] = 0.5
S0 = S(field,0.1)
Sf0 = S_field(field,0.1)[12,17]
field[12,17] = 0.3
S1 = S(field,0.1)
Sf1 = S_field(field,0.1)[12,17]
print(S1 - S0)
print(Sf1 - Sf0)
```

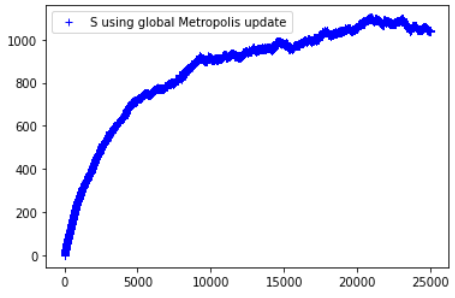
```
(327.52809672707235+0j)
(-0.5412363742706248+0j)
(-0.5412363742706198+0j)
```

In [9]:
```
def metropolis_global(phi, action, step):
    global rng
    tmp = g.real(grid)
    phi_prime = g(phi + rng.element(tmp) * step)
    S0 = action(phi)
    S1 = action(phi_prime)
    r = np.exp(S0 - S1)
    l = rng.uniform_real(min=0.0, max=1.0)
    if l < r:
        # accept
        phi @= phi_prime
        return True
    return False

field[:] = 0.0
total = 0
accepted = 0
action_list_global = []
for i in range(25000):
    if metropolis_global(field, lambda field: S(field, 1.0),0.04):
        accepted+=1
    total += 1
    action_list_global.append(S(field, 1.0).real)
    if i % 1000 == 0:
        print(i,"Acceptance rate:",accepted/total)
```

```
0 Acceptance rate: 1.0
1000 Acceptance rate: 0.5354645354645354
2000 Acceptance rate: 0.5417291354322838
3000 Acceptance rate: 0.5428190603132289
4000 Acceptance rate: 0.5546113471632091
5000 Acceptance rate: 0.5530893821235753
6000 Acceptance rate: 0.5504082652891185
7000 Acceptance rate: 0.5513498071704043
8000 Acceptance rate: 0.5504311961004874
9000 Acceptance rate: 0.551716475947117
10000 Acceptance rate: 0.5528447155284472
11000 Acceptance rate: 0.5544041450777202
12000 Acceptance rate: 0.5569535872010666
13000 Acceptance rate: 0.5564187370202293
14000 Acceptance rate: 0.556460252839083
15000 Acceptance rate: 0.5568295446970202
16000 Acceptance rate: 0.5552777951378038
17000 Acceptance rate: 0.5561437562496324
18000 Acceptance rate: 0.5571357146825177
19000 Acceptance rate: 0.558549550023683
20000 Acceptance rate: 0.559622018899055
21000 Acceptance rate: 0.5599733346031142
22000 Acceptance rate: 0.5607472387618745
23000 Acceptance rate: 0.5600626059736533
24000 Acceptance rate: 0.5603099870838715
```

In [10]:

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

ax.plot(range(len(action_list_global)), action_list_global, marker='+', ls='', c='blue', label="S using global Metropolis update")

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



In [11]:

```
def metropolis_step(phi, action, step):
    global rng
    # update even fields then odd fiels, in this way we can update
    # half of the sites with individual accept/reject at each step
    tmp = g.real(grid)
    tmp_cb = g.real(grid_eo)
    r_cb = g.real(grid_eo)
    mask_accept = g.real(grid)

    total_accept = 0
    total_step = 0

    for cb in [g.even, g.odd]:

        # p(x) = const * exp(-S(x)) -> p(x') / p(x) = exp(S(x) - S(x'))
        S_field_0 = action(phi) # wasteful to compute every time, for now fine

        # create phi_prime which only has shifts on cb sites
        tmp[:] = 0
        tmp_cb.checkerboard(cb)
        rng.element(tmp_cb)
        g.set_checkerboard(tmp, tmp_cb)
        phi_prime = g(phi + step * tmp)

        # difference of action (per site)
        sdiff = g(S_field_0 - action(phi_prime))
        g.pick_checkerboard(cb, tmp_cb, sdiff)
        pdiff = g(g.component.exp(tmp_cb))

        # on same sites draw independent numbers between 0 and 1
        rng.uniform_real(r_cb, min = 0.0, max = 1.0)

        # and locally do an accept/reject
        mask_accept_cb = g(pdiff > r_cb)
        mask_accept[:] = 0.0
        mask_accept_cb.checkerboard(cb) # in future versions of gpt, this will be inferred from >
        g.set_checkerboard(mask_accept, mask_accept_cb)

        # statistics
        total_accept += g.norm2(mask_accept)
        total_step += mask_accept_cb.grid.gsites

        # locally keep accepted changes
        phi @= g.where(mask_accept, phi_prime, phi)

    return total_accept / total_step
```

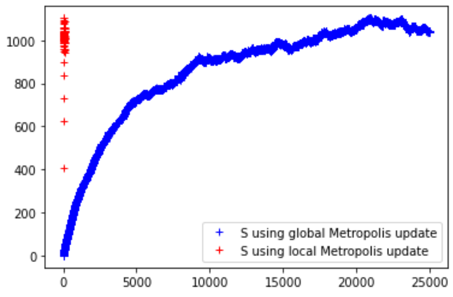
```
field[:] = 0.0
action_list = []
for s in range(100):
    acceptance_rate = metropolis_step(field, lambda field: S_field(field, 1.0), 1.6)
    action_list.append(S(field, 1.0).real)
    if s % 10 == 0:
        print("Acceptance rate of local update:",acceptance_rate)
```

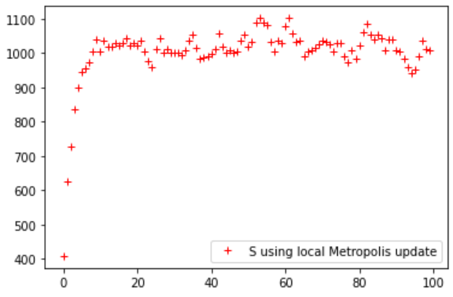
```
Acceptance rate of local update: 0.654296875
Acceptance rate of local update: 0.66748046875
Acceptance rate of local update: 0.66943359375
Acceptance rate of local update: 0.67138671875
Acceptance rate of local update: 0.6787109375
Acceptance rate of local update: 0.67333984375
Acceptance rate of local update: 0.6689453125
Acceptance rate of local update: 0.66162109375
Acceptance rate of local update: 0.6552734375
Acceptance rate of local update: 0.6611328125
```

In [ ]:

In [12]:

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

ax.plot(range(len(action_list_global)), action_list_global, marker='+', ls='', c='blue', label="S using global Metropolis update")
ax.plot(range(len(action_list)), action_list, marker='+', ls='', c='red', label="S using local Metropolis update")

plt.legend()
plt.show()

fig, ax = plt.subplots()

ax.plot(range(len(action_list)), action_list, marker='+', ls='', c='red', label="S using local Metropolis update")

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





Momentum eigenstates and two-point correlators

Since the Hamiltonian for non-interacting real scalar fields in one spatial dimension is

$$H = \sum_{k=0}^{n_s-1} w_k \left[\frac{1}{2} + \hat{n}_k\right] \tag{33}$$

with occupation number operator

$$\hat{n}_k = \hat{a}_k^\dagger \hat{a}_k \,, \tag{34}$$

we know that the eigenstates can be classified with quantum numbers $n_1, \ldots, n_{n_s-1}$ with

$$\hat{n}_k |n_1, \ldots, n_{n_s-1}\rangle = n_k |n_1, \ldots, n_{n_s-1}\rangle \,. \tag{35}$$

We call the special state with lowest energy the vaccum state

$$|0\rangle = |0, \ldots, 0\rangle \tag{36}$$

with vacuum energy E_0 already defined above. We can then study the operator

$$\phi_k^\dagger = \frac{1}{\sqrt{2\omega_k n_s}} \sum_{x=0}^{n_s-1} e^{ip_k x}\phi(x,0) \tag{37}$$
$$= \sum_{x=0}^{n_s-1} f_k(x,0)\phi(x,0) \tag{38}$$
$$= \frac{1}{2\omega_k}(a_k^\dagger + a_{-k}) \tag{39}$$

such that

$$\phi_k^\dagger|0\rangle = \frac{1}{2\omega_k}|0,\ldots,0,n_k=1,0,\ldots 0\rangle \tag{40}$$

where we used $a_k|0\rangle = 0$.

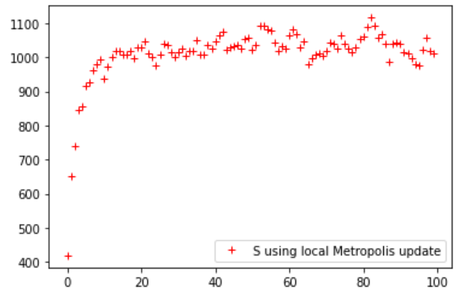
We can therefore study

$$C(t) = \langle \phi_k(t)\phi_k^\dagger(0)\rangle \tag{41}$$
$$= \frac{1}{Z}\sum_{n,m}\langle n|\phi_k|m\rangle\langle m|\phi_k^\dagger|n\rangle e^{-tE_m-(T-t)E_n} \tag{42}$$
$$\approx \sum_m \langle 0|\phi_k|m\rangle\langle m|\phi_k^\dagger|0\rangle e^{-t(E_m-E_0)} \tag{43}$$
$$\approx \sum_m \langle 0|\phi_k|m\rangle\langle m|\phi_k^\dagger|0\rangle e^{-t(E_m-E_0)} \tag{44}$$
$$\approx |\langle\phi_k|\phi_k\rangle|^2 e^{-t(E_{n_k=1}-E_0)}\,, \tag{45}$$

where we considered the limits $1 \ll t \ll T$.

In [13]:
```
rng = g.random("seed13")
field[:] = 0.0
action_list = []
m0 = 0.2

# first thermalize theory for new m0
for s in range(100):
    acceptance_rate = metropolis_step(field, lambda field: S_field(field, m0), 2.0)
    action_list.append(S(field, m0).real)
    if s % 10 == 0:
        print("Acceptance rate of local update:",acceptance_rate)

fig, ax = plt.subplots()

ax.plot(range(len(action_list)), action_list, marker='+', ls='', c='red', label="S using local Metropolis update")

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

```
GPT :      62.218938 s : Initializing gpt.random(seed13,vectorized_ranlux24_389_64) took 0.000290871 s
Acceptance rate of local update: 0.60791015625
Acceptance rate of local update: 0.62109375
Acceptance rate of local update: 0.64208984375
Acceptance rate of local update: 0.62255859375
Acceptance rate of local update: 0.64453125
Acceptance rate of local update: 0.6376953125
Acceptance rate of local update: 0.63525390625
Acceptance rate of local update: 0.62744140625
Acceptance rate of local update: 0.6220703125
Acceptance rate of local update: 0.6357421875
```



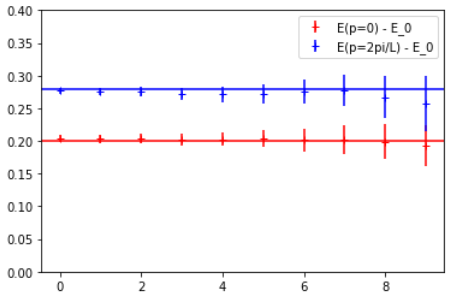
In [14]:
```
configurations = []

for s in range(5000):
    for sub_steps in range(5):
        acceptance_rate = metropolis_step(field, lambda field: S_field(field, m0), 2.0)
    configurations.append(g.copy(field))
    if s % 1000 == 0:
        print(s)
```

```
0
1000
2000
3000
4000
```

In [15]:
```
def correlate(field):
    phit = g.slice(field,1)
    Nt=len(phit)
    return np.array([ sum([ phit[s].conjugate()*phit[(s+t)%Nt] for s in range(Nt) ])/Nt for t in range(Nt) ])

def two_point_for_momentum(p):
    mom_phase = g.exp_ixp(np.array([p,0]))
    correlators = [ correlate(g(mom_phase*conf)) for conf in configurations ]
    return correlators

c0 = two_point_for_momentum(0)
c1 = two_point_for_momentum(2.*np.pi/L[0])
```

In [16]:

```
def jackknife_variance_estimator(f, x):
    N = len(x)
    X = sum(x) / N
    mean = f(X)
    return sum([(f((N*X - x[j])/(N-1)) - mean)**2 for j in range(N)])*(N-1)/N

def bin_data(res, bin_size):
    assert len(res) % bin_size == 0
    n = len(res) // bin_size
    return [ sum(res[bin_size*i:bin_size*(i+1)])/bin_size for i in range(n) ]

def time_parity_average(C):
    return [0.5*np.roll(x[::-1],1) + 0.5*x for x in C]

bc0 = time_parity_average(bin_data(c0, 100))
bc1 = time_parity_average(bin_data(c1, 100))

def emp(C):
    return np.array([np.log(C[t].real/C[t+1].real) for t in range(10)])

c0_mean = np.mean(bc0, axis=0)
emp0_mean = emp(c0_mean)
emp0_err = jackknife_variance_estimator(emp, bc0)**0.5

c1_mean = np.mean(bc1, axis=0)
emp1_mean = emp(c1_mean)
emp1_err = jackknife_variance_estimator(emp, bc1)**0.5

fig, ax = plt.subplots()

plt.ylim([0,0.4])
ax.errorbar(range(len(emp0_mean)), emp0_mean, emp0_err, marker='+', ls='', c='red', label="E(p=0) - E_0")
ax.errorbar(range(len(emp1_mean)), emp1_mean, emp1_err, marker='+', ls='', c='blue', label="E(p=2pi/L) - E_0")

plt.axhline(y=m0, color='red', linestyle='-')
plt.axhline(y=(0.2**2. + (2.*np.sin(np.pi/L[0]))**2.)**0.5, color='blue', linestyle='-')

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



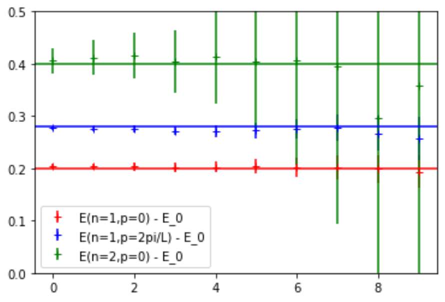
We can also create two particles, say, in the $k = 0$ state by studying

$$C(t) = \langle (\phi_k(t))^2 (\phi_k^\dagger(0))^2 \rangle - \langle (\phi_k(t))^2 \rangle^2 \,. \tag{46}$$

Homework: Show how one can extract a state with occupation number two from this correlator.

In [17]:

```
def correlate2(field):
    phit = g.slice(field,1)
    Nt=len(phit)
    return np.array([ sum([ phit[s].conjugate()**2*phit[(s+t)%Nt]**2 for s in range(Nt) ])/Nt for t in range(Nt) ])

def two_point_for_momentum2(p):
    mom_phase = g.exp_ixp(np.array([p,0]))
    correlators = [ correlate2(g(mom_phase*conf)) for conf in configurations ]
    return correlators

c2 = two_point_for_momentum2(0)

bc2 = time_parity_average(bin_data(c2, 100))

# variance of this is negligible, so treat it as a constant
c_const = np.ones(len(c0_mean))*c0_mean[0]**2.

c2_mean = np.mean(bc2, axis=0)
emp2_mean = emp(c2_mean - c_const)
emp2_err = jackknife_variance_estimator(lambda c2: emp(c2 - c_const), bc2)**0.5

fig, ax = plt.subplots()

plt.ylim([0,0.5])
ax.errorbar(range(len(emp0_mean)), emp0_mean, emp0_err, marker='+', ls='', c='red', label="E(n=1,p=0) - E_0")
ax.errorbar(range(len(emp1_mean)), emp1_mean, emp1_err, marker='+', ls='', c='blue', label="E(n=1,p=2pi/L) - E_0")
ax.errorbar(range(len(emp2_mean)), emp2_mean, emp2_err, marker='+', ls='', c='green', label="E(n=2,p=0) - E_0")

plt.axhline(y=m0, color='red', linestyle='-')
plt.axhline(y=(0.2**2. + (2.*np.sin(np.pi/32))**2.)**0.5, color='blue', linestyle='-')
plt.axhline(y=2*0.2, color='green', linestyle='-')


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

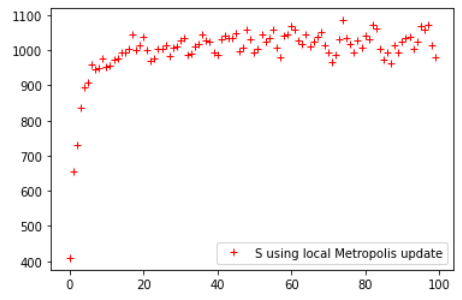


Next, we can add a small interaction term and observe how the states shift. Note that even the one-particle state can shift and m_0 is in general no longer its energy!

In [19]:

```
rng = g.random("seed13")
action_list = []
m0 = 0.2
lam = 0.005

def S_ia(phi, m0, lam):
    action = 0.5*g.norm2(phi)*m0**2 + lam * g.sum(g.adj(phi)*phi*g.adj(phi)*phi)
    for mu in range(2):
        action -= 0.5 * g.inner_product(phi, g.cshift(phi,mu,1) + g.cshift(phi,mu,-1) - 2.*phi)
    return action

def S_field_ia(phi, m0, lam):
    # gather in afld[x] everything that changes if phi[x] changes
    afld = g(0.5 * g.adj(phi)*phi*(m0**2 + 4.0))
    afld += lam * g.adj(phi)*phi*g.adj(phi)*phi
    for mu in range(2):
        afld -= g.adj(phi) * (g.cshift(phi,mu,1) + g.cshift(phi,mu,-1))
    return afld

# test change in single point
rng.element(field)
field[12,17] = 0.5
S0 = S_ia(field,m0,lam)
Sf0 = S_field_ia(field,m0,lam)[12,17]
field[12,17] = 0.3
S1 = S_ia(field,m0,lam)
Sf1 = S_field_ia(field,m0,lam)[12,17]
print(S1 - S0)
print(Sf1 - Sf0)



# first thermalize theory for new m0
field[:] = 0.0
for s in range(100):
    acceptance_rate = metropolis_step(field, lambda field: S_field_ia(field, m0, lam), 3.0)
    action_list.append(S_ia(field, m0, lam).real)
    if s % 10 == 0:
        print("Acceptance rate of local update:",acceptance_rate)

fig, ax = plt.subplots()

ax.plot(range(len(action_list)), action_list, marker='+', ls='', c='red', label="S using local Metropolis update")

plt.legend()
plt.show()

configurations_ia = []

for s in range(5000):
    for sub_steps in range(5):
        acceptance_rate = metropolis_step(field, lambda field: S_field_ia(field, m0, lam), 3.0)
    configurations_ia.append(g.copy(field))
    if s % 1000 == 0:
        print(s)
```

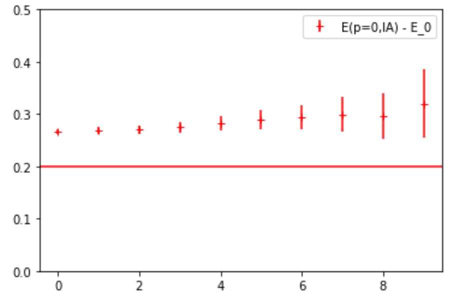
```
GPT :     442.801170 s : Initializing gpt.random(seed13,vectorized_ranlux24_389_64) took 0.000324965 s
(-0.543908374270643+0j)
(-0.5439083742706198+0j)
Acceptance rate of local update: 0.4384765625
Acceptance rate of local update: 0.48779296875
Acceptance rate of local update: 0.49755859375
Acceptance rate of local update: 0.4951171875
Acceptance rate of local update: 0.49609375
Acceptance rate of local update: 0.5126953125
Acceptance rate of local update: 0.49462890625
Acceptance rate of local update: 0.49365234375
Acceptance rate of local update: 0.494140625
Acceptance rate of local update: 0.4970703125
```



```
0
1000
2000
3000
4000
```

In [20]:

```
def two_point_for_momentum_ia(p):
    mom_phase = g.exp_ixp(np.array([p,0]))
    correlators = [ correlate(g(mom_phase*conf)) for conf in configurations_ia ]
    return correlators

c0_ia = two_point_for_momentum_ia(0)
bc0_ia = time_parity_average(bin_data(c0_ia, 100))

c0_ia_mean = np.mean(bc0_ia, axis=0)
emp0_ia_mean = emp(c0_ia_mean)
emp0_ia_err = jackknife_variance_estimator(emp, bc0_ia)**0.5

fig, ax = plt.subplots()

plt.ylim([0,0.5])
ax.errorbar(range(len(emp0_ia_mean)), emp0_ia_mean, emp0_ia_err, marker='+', ls='', c='red', label="E(p=0,IA) - E_0")

plt.axhline(y=m0, color='red', linestyle='-')

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



One can then also measure the energy of two particles in the interacting theory. They will no longer be twice as energetic as a single particle due to the interaction energy. By studying these interaction energies at different volumes, one can learn about scattering amplitudes of particles (see the Lüscher formalism https://doi.org/10.1016/0550-3213(91)90366-6).

Field theories in higher dimensions

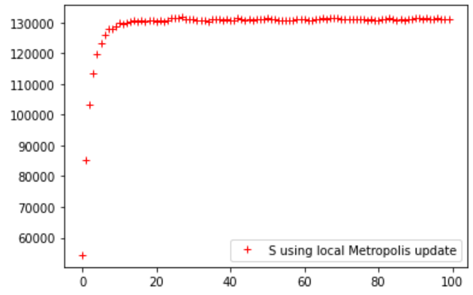
It is straightforward to modify the presented code to simulations in higher dimensions. Below for a 3+1d interacting real scalar field theory.

In [21]:

```
L = [16,16,16,64]
print(L[0]*L[1]*L[2]*L[3])
grid = g.grid(L, g.double)
grid_eo = g.grid(L, g.double, g.redblack)

field = g.real(grid)
field[:] = 0

rng = g.random("seed13")
field[:] = 0.0
action_list = []
m0 = 0.2
lam = 0.005

def S_4d(phi, m0, lam):
    action = 0.5*g.norm2(phi)*m0**2 + lam * g.sum(g.adj(phi)*phi*g.adj(phi)*phi)
    for mu in range(4):
        action -= 0.5 * g.inner_product(phi, g.cshift(phi,mu,1) + g.cshift(phi,mu,-1) - 2.*phi)
    return action

def S_field_4d(phi, m0, lam):
    # gather in afld[x] everything that changes if phi[x] changes
    afld = g(0.5 * g.adj(phi)*phi*(m0**2 + 8.0))
    afld += lam * g.adj(phi)*phi*g.adj(phi)*phi
    for mu in range(4):
        afld -= g.adj(phi) * (g.cshift(phi,mu,1) + g.cshift(phi,mu,-1))
    return afld

# first thermalize theory for new m0
```

```python
for s in range(100):
    acceptance_rate = metropolis_step(field, lambda field: S_field_4d(field, m0, lam), 2.0)
    action_list.append(S_4d(field, m0, lam).real)
    if s % 10 == 0:
        print("Acceptance rate of local update:",acceptance_rate)

fig, ax = plt.subplots()

ax.plot(range(len(action_list)), action_list, marker='+', ls='', c='red', label="S using local Metropolis update")

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

```
262144
GPT :     667.002744 s : Initializing gpt.random(seed13,vectorized_ranlux24_389_64) took 0.000406027 s
Acceptance rate of local update: 0.4446144104003906
Acceptance rate of local update: 0.5132102966308594
Acceptance rate of local update: 0.5131492614746094
Acceptance rate of local update: 0.5140342712402344
Acceptance rate of local update: 0.5136299133300781
Acceptance rate of local update: 0.5135154724121094
Acceptance rate of local update: 0.512176513671875
Acceptance rate of local update: 0.5137062072753906
Acceptance rate of local update: 0.5137748718261719
Acceptance rate of local update: 0.5127792358398438
```


S using local Metropolis update