Chapter 6: gauge theories on a lattice

In this chapter we explore how we can construct a field theory on a lattice that keeps the gauge symmetry intact.

Covariant derivatives in the continuum

In the last chapter we considered generators $G(x)$ of a gauge symmetry group, which in the case of SU(3) are traceless Hermitian matrices ($G^\dagger = G$, $\mathrm{Tr} G = 0$). We found that we can define covariant derivatives D_μ that transformed as

$$D_\mu \to [1 + iG(x)] D_\mu [1 - iG(x)] \tag{1}$$

under infinitesimal gauge transformations. Finite gauge transformations can then be written as

$$D_\mu \to V(x) D_\mu V(x)^\dagger \tag{2}$$

with

$$V(x) = \exp(iG(x))\,. \tag{3}$$

These then allowed for the definition of a gauge invariant kinetic term of a fermionic field $\psi(x)$ that transforms as

$$\psi(x) \to V(x)\psi(x)\,, \tag{4}$$
$$\bar\psi(x) \to \bar\psi(x) V(x)^\dagger \tag{5}$$

using

$$\mathcal{L} = \bar\psi i \gamma^\mu D_\mu \psi\,. \tag{6}$$

Discrete derivatives on a lattice

We now consider fields $\psi(x)$ that live on a discrete lattice with $x \in \Lambda$ and $\Lambda = \{an | n \in \mathbb{Z}^d\}$ with lattice dimension d and lattice spacing a. Then an approximations of the partial derivatives in direction μ can be writte, e.g., as

$$a\overrightarrow{\partial}_\mu \psi(x) = \psi(x + a\hat\mu) - \psi(x)\,, \tag{7}$$
$$a\overleftarrow{\partial}_\mu \psi(x) = \psi(x) - \psi(x - a\hat\mu) \tag{8}$$

with unit vectors $\hat\mu$ in direction μ. If we imagine ψ being a sufficiently differentiable field in $\mathbb{R}^d$ instead, we find

$$\overrightarrow{\partial}_\mu \psi(x) = \frac{1}{a}(\psi(x + a\hat\mu) - \psi(x)) = \partial_\mu \psi(x) + O(a)\,, \tag{9}$$
$$\overleftarrow{\partial}_\mu \psi(x) = \frac{1}{a}(\psi(x) - \psi(x - a\hat\mu) = \partial_\mu \psi(x) + O(a) \tag{10}$$

and if we take a symmetric combination

$$\overleftrightarrow{\partial}_\mu = \frac{1}{2}(\overleftarrow{\partial}_\mu + \overrightarrow{\partial}_\mu) \tag{11}$$

we find

$$\overleftrightarrow{\partial}_\mu \psi(x) = \frac{1}{2a}(\psi(x + a\hat\mu) - \psi(x - a\hat\mu)) = \partial_\mu \psi(x) + O(a^2)\,. \tag{12}$$

Homework: Show this for the symmetric derivative by performing a corresponding Taylor expansion.

Covariant derivatives on a lattice

We have just seen that on a lattice we need to approximate derivatives with finite differences of fields shifted in discrete steps on the lattice. We can make this more explicit by defining shift operators

$$S_\mu^+ \psi(x) = \psi(x + a\hat\mu)\,, \tag{13}$$
$$S_\mu^- \psi(x) = \psi(x - a\hat\mu) \tag{14}$$

such that

$$\overleftrightarrow{\partial}_\mu \psi(x) = \frac{1}{2a}(S_\mu^+ - S_\mu^-)\psi(x) \tag{15}$$

If we assume again that under a gauge transformation each field transforms as

$$\psi(x) \to V(x)\psi(x) \tag{16}$$

the partial derivatives $\overleftrightarrow{\partial}_\mu \psi(x)$ again would not have a well-defined behavior under gauge transformations. What we would need is a covariant shift operation

$$C_\mu^+ \psi(x) = U_\mu(x) \psi(x + a\hat\mu) \tag{17}$$

which should obey

$$C_\mu^+ \psi(x) \to V(x) C_\mu^+ \psi(x) \tag{18}$$

under a gauge transformation. This can be achieved if the $U_\mu(x)$ above were to transform as

$$U_\mu(x) \to V(x) U_\mu(x) V(x+a\hat\mu)^\dagger \tag{19}$$

under gauge transformations. Then we could also define a covariant backwards shift operation

$$C_\mu^- \psi(x) = U_\mu^\dagger(x-a\hat\mu)\psi(x-a\hat\mu) \tag{20}$$

that would transform as

$$C_\mu^- \psi(x) \to V(x) C_\mu^- \psi(x) \tag{21}$$

under gauge transformations.

Then by replacing all shifts by covariant shifts, we can construct a **covariant derivative**

$$D_\mu \psi(x) = \frac{1}{2a}(C_\mu^+ - C_\mu^-)\psi(x) = \frac{1}{2a}\left(U_\mu(x)\psi(x+a\hat\mu) - U_\mu^\dagger(x-a\hat\mu)\psi(x-a\hat\mu)\right) . \tag{22}$$

As operators, we can write

$$C_\mu^+ = U_\mu(x) S_\mu^+ \,, \tag{23}$$
$$C_\mu^- = S_\mu^- U_\mu^\dagger(x) \,. \tag{24}$$

We can also consider the $C_\mu^\pm$ as **parallel transporters** of the gauge group.

The parallel transport matrices $U_\mu(x)$ therefore seem to play the role of transporting a field ψ between x and $x+a\hat\mu$. It is therefore only natural to assume geometrically that $U_\mu(x)$ corresponds to a **link** on the lattice between those two sites.

If we furthermore make the ansatz

$$U_\mu(x) = \left(1 + ig\frac{a}{n}A_\mu(x)\right)\left(1 + ig\frac{a}{n}A_\mu\left(x + \frac{a}{n}\hat\mu\right)\right)\cdots\left(1 + ig\frac{a}{n}A_\mu\left(x + \frac{a(n-1)}{n}\hat\mu\right)\right) \tag{25}$$
$$\overset{n\to\infty}{=} \mathcal{P}\exp\left(ig\int_0^a ds A_\mu(x+s\hat\mu)\right) = \exp\left(igaA_\mu\left(x+\frac{1}{2}a\hat\mu\right) + O(a^3)\right) \tag{26}$$

with gauge fields A_μ living on the links, we find

$$D_\mu\psi(x) = (\partial_\mu + igA_\mu(x))\psi(x) + O(a^2) \,. \tag{27}$$

The operator $\mathcal{P}$ is the **path-ordering** operator that arises from the limit of multiplying the individual factors.

Note that g is the coupling constant that we introduced at the end of the previous chapter.

Homework: Show this equation by explicit Taylor expansion in a.

Pure gauge theory

Consider the smallest possible parallel transport around a $1x1$ square in the μ-ν plane, the **plaquette**,

$$U_{\mu\nu}(x) = U_\mu(x) U_\nu(x+a\hat\mu) U_\mu^\dagger(x+a\hat\nu) U_\nu^\dagger(x) \tag{28}$$

transforming to

$$U_{\mu\nu}(x) \to V(x) U_{\mu\nu}(x) V(x)^\dagger \,. \tag{29}$$

We can then create a gauge-invariant contribution to an action by taking the trace of $U_{\mu\nu}$. Before doing this, however, let us consider what this corresponds to in terms of gauge fields A_μ. The plaquette is also the simplest of the **Wilson loops**, i.e., parallel transports around a closed loop.

If we use the Baker-Campbell-Hausdorff formula

$$\exp(aA)\exp(aB) = \exp\left(aA + aB + \frac{1}{2}a^2[A,B] + O(a^3)\right) \tag{30}$$

and combine the terms to a single exponential, we find

$$U_{\mu\nu}(x) = \exp\left(ia^2 g F_{\mu\nu}(x) + O(a^3)\right) \tag{31}$$

We can then consider the gauge-invariant trace for a gauge group SU(N_c)

$$\mathrm{Tr}\, U_{\mu\nu}(x) = N_c + ia^2 g \mathrm{Tr}\, F_{\mu\nu}(x) - \frac{1}{2}a^4 g^2 \mathrm{Tr}\, (F_{\mu\nu}(x))^2 + O(a^5) \tag{32}$$

and

$$\mathrm{Tr}\, \left(1 - U_{\mu\nu}(x)\right) = -ia^2 g \mathrm{Tr}\, F_{\mu\nu}(x) + \frac{1}{2}a^4 g^2 \mathrm{Tr}\, (F_{\mu\nu}(x))^2 + O(a^5) \,. \tag{33}$$

Next we take the real part to create a real-valued contribution to the action and use that

$$F_{\mu\nu}^\dagger = F_{\mu\nu} \tag{34}$$

such that

$$\mathrm{Re}\, \mathrm{Tr}\, \left(1 - U_{\mu\nu}(x)\right) = \frac{1}{2}a^4 g^2\, \mathrm{Tr}\, (F_{\mu\nu}(x))^2 + O(a^5) \,, \tag{35}$$

where no sum over repeated indices is implied. In order to recover the continuum action at leading order, we therefore need to divide by g^2 and sum over $x \in \Lambda$, μ and ν. This yields the **Wilson gauge action**

$$S_W = \frac{1}{g^2} \sum_x \sum_{\mu,\nu=0}^{d-1} \mathrm{Re}\,\mathrm{Tr}\,\left(1 - U_{\mu\nu}(x)\right) = \frac{1}{2} a^4 \sum_x \sum_{\mu,\nu=0}^{d-1} \mathrm{Tr}\,(F_{\mu\nu}(x))^2 + O(a^5)\,, \tag{36}$$

where d is the space-time dimensionality of the theory. For gauge group SU(N_c) in the fundamental representation, we customarily define

$$S_W = \frac{1}{2}\beta \sum_x \sum_{\mu,\nu=0}^{d-1} (1 - P_{\mu\nu}(x)) = \beta \sum_x \sum_{\mu<\nu} (1 - P_{\mu\nu}(x)) \tag{37}$$

with scalar plaquette

$$P_{\mu\nu}(x) = \frac{1}{N_c} \mathrm{Re}\,\mathrm{Tr}\, U_{\mu\nu}(x) \tag{38}$$

and

$$\beta = \frac{2N_c}{g^2}\,. \tag{39}$$

Note that we used

$$U_{\mu\nu}(x)^\dagger = U_\nu(x) U_\mu(x + a\hat{\nu}) U_\nu(x + a\hat{\mu})^\dagger U_\mu(x)^\dagger = U_{\nu\mu}(x) \tag{40}$$

to replace the sum over μ, ν with a sum over $\mu < \nu$.

Improved gauge theory

We can show that also other Wilson loops locally look like the field-strength tensor $F_{\mu\nu}$, albeit with different discretization errors. By taking appropriate combinations of Wilson loops, we can create an improved gauge action which reduces the discretization errors to a higher power in a. At this point, we could already determine the classically optimal combination of, e.g., the $1x1$ and $2x1$ rectangle terms, however, we will later see that quantum effects can disturb this cancellation. We will revisit this point at a later time.

Path integral measure

One can show that for finite groups $G = \{g_1, \ldots, g_n\}$ that for any element $g_i \in G$, the sequence $g_i g_0, g_i g_1, \ldots, g_i g_n$ and the sequence $g_0 g_i, g_1 g_i, \ldots, g_n g_i$ are just reorderings of the sequence $g_0, \ldots, g_n$. Therefore averaging a function f over all elements of a finite group satisfies

$$\frac{1}{n} \sum_g f(g) = \frac{1}{n} \sum_g f(g_i g) = \frac{1}{n} \sum_g f(g g_i)\,. \tag{41}$$

For continuous Lie groups that we consider in this chapter as gauge groups one can define the **Haar integration measure** $d[U]$ that satisfies

$$\int d[U] f(U) = \int d[U] f(VU) = \int d[U] f(UV) \tag{42}$$

for group elements U and V. Equivalently

$$d[VU] = d[U] = d[UV] \tag{43}$$

for each group element V. Using such an integration measure, the integral over link variables $U_\mu(x)$ then also is gauge invariant since

$$d[V(x) U_\mu(x) V(x + a\hat{\mu})^\dagger] = d[U_\mu(x)]\,. \tag{44}$$

If the action and integral measure is explicitly invariant under our gauge transformation, then the quantized theory as defined by the path integral is also explicitly gauge invariant. This is crucial since the measure may in principle violate symmetries of the action or in other words, a quantized theory may exhibit lower symmetries than the action itself has.

In a later discussion we will give a procedure to derive the Haar measure for a given parametrization of a group manifold. For now, we just observe that in a stochastic sampling of the integral all we have to follow is to pick random elements V of the gauge group and to consistently multiply them from the left or right to the existing links $U_\mu(x)$ to update them. As long as we make sure that any group element can be reached by a product of these random V, the construction explicitly follows the Haar integration measure as defined by its invariance properties. For completeness, we note that the Haar measure is only unique up to a constant factor that can be fixed, e.g., by requiring $\int d[U] = 1$.

Markov chains for pure gauge theories

For pure gauge theories, there are various competitive algorithms to generate a Markov chain:

- The first is a **local metropolis** using a checkerboarding strategy as we have done before for scalar field theory.
- Another is to explicitly parametrize the Haar measure for SU(2) subgroups of the SU(N_c) gauge group and to draw such SU(2) matrices directly from the correct probability distribution. This is the so-called SU(2)-subgroup **heatbath** algorithm. We will discuss it in detail in the next lecture.
- We could also integrate a stochastic differential equation, called a **Langevin** integrator, and show explicitly that the correct stationary distribution is found. We will also discuss this method in the next lecture.
- Finally, one can also use a combination of molecular dynamics evolution with random momentum sampling, the **hybrid monte carlo** method. We will dedicate an entire chapter to this method due to its importance in state-of-the-art calculations.

In the following, we first demonstrate the ensemble generation with GPT using some of these algorithms.

First using the local Metropolis:

In [3]:
```
import gpt as g;
```

```
L = [8, 8, 8, 8]
grid = g.grid(L, g.single)
grid_eo = g.grid(L, g.single, g.redblack)

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

# red/black mask
mask_rb = g.complex(grid_eo)
mask_rb[:] = 1

# full mask
mask = g.complex(grid)

# simple plaquette action
def staple(U, mu):
    st = g.lattice(U[0])
    st[:] = 0
    Nd = len(U)
    for nu in range(Nd):
        if mu != nu:
            st += g.qcd.gauge.staple(U, mu, nu) / U[0].otype.Nc
    return st

# g.default.push_verbose("local_metropolis", True) # activaties display of acceptance rate
markov = g.algorithms.markov.local_metropolis(rng, step_size=0.5)
beta = 5.5

plaquette_local_metropolis = []
for it in range(1000):
    plaq = g.qcd.gauge.plaquette(U)
    plaquette_local_metropolis.append(plaq)
    if it % 50 == 0:
        g.message(f"Local metropolis {it} has P = {plaq}")
    for cb in [g.even, g.odd]:
        mask[:] = 0
        mask_rb.checkerboard(cb)
        g.set_checkerboard(mask, mask_rb)

        for mu in range(Nd):
            st = g.eval(beta * staple(U, mu))
            markov(U[mu], st, mask)
```

```
GPT :     113.770477 s : Initializing gpt.random(test,vectorized_ranlux24_24_64) took 4.60148e-05 s
GPT :     113.785935 s : Local metropolis 0 has P = 1.0
GPT :     118.091342 s : Local metropolis 50 has P = 0.6232448286480373
GPT :     122.690226 s : Local metropolis 100 has P = 0.558342350853814
GPT :     126.951510 s : Local metropolis 150 has P = 0.5412292811605666
GPT :     131.852604 s : Local metropolis 200 has P = 0.5315725008646647
GPT :     136.840365 s : Local metropolis 250 has P = 0.5241931941774156
GPT :     142.993895 s : Local metropolis 300 has P = 0.5202440818150839
GPT :     147.940617 s : Local metropolis 350 has P = 0.5177053146892124
GPT :     152.788150 s : Local metropolis 400 has P = 0.5131866865687901
GPT :     157.826834 s : Local metropolis 450 has P = 0.5104659663306342
GPT :     162.208752 s : Local metropolis 500 has P = 0.5131155252456665
GPT :     166.561504 s : Local metropolis 550 has P = 0.5085434781180488
GPT :     171.248844 s : Local metropolis 600 has P = 0.5039165417353312
GPT :     175.708371 s : Local metropolis 650 has P = 0.506249725818634
GPT :     180.486332 s : Local metropolis 700 has P = 0.5023225479655796
GPT :     185.004515 s : Local metropolis 750 has P = 0.5003030101458231
GPT :     189.150137 s : Local metropolis 800 has P = 0.4997691445880466
GPT :     193.177941 s : Local metropolis 850 has P = 0.5011237727271186
GPT :     197.519407 s : Local metropolis 900 has P = 0.5038102865219116
GPT :     201.544591 s : Local metropolis 950 has P = 0.5002213915189108
```

Next we consider the Heatbath algorithm.

In [7]:
```
g.default.push_verbose("su2_heat_bath", False)
markov = g.algorithms.markov.su2_heat_bath(rng)
U = g.qcd.gauge.unit(grid)
plaquette_heatbath = []
for it in range(500):
    plaq = g.qcd.gauge.plaquette(U)
    plaquette_heatbath.append(plaq)
    if it % 50 == 0:
        g.message(f"SU(2)-subgroup heatbath {it} has P = {plaq}")
    for cb in [g.even, g.odd]:
        mask[:] = 0
        mask_rb.checkerboard(cb)
        g.set_checkerboard(mask, mask_rb)

        for mu in range(Nd):
            st = g.eval(beta * staple(U, mu))
            markov(U[mu], st, mask)
```

```
GPT :     300.293935 s : SU(2)-subgroup heatbath 0 has P = 1.0
GPT :     313.256864 s : SU(2)-subgroup heatbath 50 has P = 0.5022712548573812
GPT :     326.851409 s : SU(2)-subgroup heatbath 100 has P = 0.4951641890737746
GPT :     341.379575 s : SU(2)-subgroup heatbath 150 has P = 0.49848774406645036
GPT :     356.706945 s : SU(2)-subgroup heatbath 200 has P = 0.5004462268617418
GPT :     373.774078 s : SU(2)-subgroup heatbath 250 has P = 0.49490469694137573
GPT :     389.923751 s : SU(2)-subgroup heatbath 300 has P = 0.49955056111017865
GPT :     405.860566 s : SU(2)-subgroup heatbath 350 has P = 0.49755699104732937
GPT :     420.428378 s : SU(2)-subgroup heatbath 400 has P = 0.499170528517829
GPT :     435.301122 s : SU(2)-subgroup heatbath 450 has P = 0.5017324553595649
```

Finally, we demonstrate the Langevin algorithm.

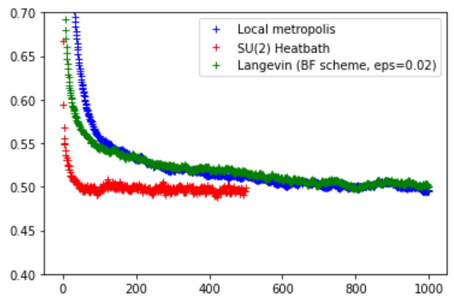
In [9]:
```
U = g.qcd.gauge.unit(grid)
w = g.qcd.gauge.action.wilson(beta)
l = g.algorithms.markov.langevin_bf(rng, epsilon=0.02)
plaquette_langevin = []
for it in range(1000):
    l(U, w)
    plaq = g.qcd.gauge.plaquette(U)
    plaquette_langevin.append(plaq)
    if it % 50 == 0:
        g.message(f"Langevin_bf(eps=0.02) {it} has P = {plaq}")
```

```
GPT :     511.601426 s : Langevin_bf(eps=0.02) 0 has P = 0.913337786992391
GPT :     514.370879 s : Langevin_bf(eps=0.02) 50 has P = 0.566335055563185
GPT :     517.300690 s : Langevin_bf(eps=0.02) 100 has P = 0.5438746478822496
GPT :     520.499172 s : Langevin_bf(eps=0.02) 150 has P = 0.5410566197501289
GPT :     523.587544 s : Langevin_bf(eps=0.02) 200 has P = 0.533349891503652
GPT :     527.598893 s : Langevin_bf(eps=0.02) 250 has P = 0.5256600843535529
GPT :     532.062941 s : Langevin_bf(eps=0.02) 300 has P = 0.5235162112447951
GPT :     535.788070 s : Langevin_bf(eps=0.02) 350 has P = 0.5187506278355917
GPT :     541.385091 s : Langevin_bf(eps=0.02) 400 has P = 0.5197881857554117
GPT :     547.443904 s : Langevin_bf(eps=0.02) 450 has P = 0.5210425655047098
GPT :     552.004728 s : Langevin_bf(eps=0.02) 500 has P = 0.5160737898614671
GPT :     555.507547 s : Langevin_bf(eps=0.02) 550 has P = 0.5104968547821045
GPT :     559.059208 s : Langevin_bf(eps=0.02) 600 has P = 0.5076582895384895
GPT :     562.807271 s : Langevin_bf(eps=0.02) 650 has P = 0.5071470008956062
GPT :     566.821152 s : Langevin_bf(eps=0.02) 700 has P = 0.5031868351830376
GPT :     570.149039 s : Langevin_bf(eps=0.02) 750 has P = 0.5077733132574294
GPT :     573.409066 s : Langevin_bf(eps=0.02) 800 has P = 0.498729235596127
GPT :     576.775995 s : Langevin_bf(eps=0.02) 850 has P = 0.5024757583936056
GPT :     579.928553 s : Langevin_bf(eps=0.02) 900 has P = 0.5047208666801453
GPT :     583.567940 s : Langevin_bf(eps=0.02) 950 has P = 0.5034748845630223
```

In [16]:
```
from matplotlib import pyplot as plt
fig, ax = plt.subplots()

plt.ylim([0.4,0.7])
ax.plot(range(len(plaquette_local_metropolis)), plaquette_local_metropolis, marker='+', ls='', c='blue', label="Local metropolis")
ax.plot(range(len(plaquette_heatbath)), plaquette_heatbath, marker='+', ls='', c='red', label="SU(2) Heatbath")
ax.plot(range(len(plaquette_langevin)), plaquette_langevin, marker='+', ls='', c='green', label="Langevin (BF scheme, eps=0.02)")

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



Homework: Measure the average 1x1 and 2x1 Wilson loops for $\beta = 5.5$. Write your own implementation based on the **g.cshift** function and compare to **g.qcd.gauge.rectangle**. See this test for an explanation of the latter function.

Haar integration for SU(2)

Before explaining in detail how the heat bath algorithm works, we first discuss the Haar integration for SU(2) in some detail.

Haar measure for explicit parametrization

If we parametrize a general 2x2 matrix

$$U = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \tag{45}$$

its inverse (if it exists) can be written as

$$U^{-1} = \frac{1}{\det(U)} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \tag{46}$$

such that for $U \in \mathrm{SU}(2)$ we find

$$U^{-1} = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} = U^\dagger = \begin{pmatrix} a^* & c^* \\ b^* & d^* \end{pmatrix} \tag{47}$$

and therefore

$$d = a^* , \qquad c = -b^* . \tag{48}$$

A general SU(2) matrix can thus be written as

$$U = \begin{pmatrix} a & b \\ -b^* & a^* \end{pmatrix} \tag{49}$$

with

$$|a|^2 + |b|^2 = 1\,. \tag{50}$$

can therefore also be expressed as

$$U = \Sigma \cdot x \tag{51}$$

with $a = x_0 + ix_3$, $b = x_2 + ix_1$ and

$$\Sigma_0 = 1\,, \tag{52}$$
$$\Sigma_i = i\sigma_i \tag{53}$$

and Pauli matrices

$$\sigma_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\,, \qquad \sigma_2 = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}\,, \qquad \sigma_3 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\,. \tag{54}$$

such that

$$\det(U) = |x|^2 = 1\,. \tag{55}$$

This parametrization allows for a straightforward definition of a Haar integration measure. It is straightforward to show that for

$$V = \Sigma \cdot y\,, \tag{56}$$
$$UV = \Sigma \cdot z\,, \tag{57}$$

we have

$$\det\left(\frac{\partial z_\mu}{\partial x_\nu}\right) = \det(V)^2 = 1 \tag{58}$$

and therefore

$$d[U] = dx_0 dx_1 dx_2 dx_3 \delta(|x| - 1) \tag{59}$$

is a Haar integration measure. For concreteness, we can transform this to spherical coordinates with

$$x_0 = \cos(\varphi_0)\,, \tag{60}$$
$$x_1 = \sin(\varphi_0)\cos(\varphi_1)\,, \tag{61}$$
$$x_2 = \sin(\varphi_0)\sin(\varphi_1)\cos(\varphi_2)\,, \tag{62}$$
$$x_3 = \sin(\varphi_0)\sin(\varphi_1)\sin(\varphi_2)\,, \tag{63}$$

and write

$$\int d[U] f(U) = \int_0^\pi d\varphi_0 \int_0^\pi d\varphi_1 \int_0^{2\pi} d\varphi_2 \sin(\varphi_0)^2 \sin(\varphi_1) f(U)\,. \tag{64}$$

Therefore we obtain the standard normalization

$$\int d[U] = 1 \tag{65}$$

for

$$\int d[U] f(U) = \frac{1}{2\pi^2} \int_0^\pi d\varphi_0 \int_0^\pi d\varphi_1 \int_0^{2\pi} d\varphi_2 \sin(\varphi_0)^2 \sin(\varphi_1) f(U) \tag{66}$$

or

$$d[U] = \frac{1}{2\pi^2} dx_0 dx_1 dx_2 dx_3 \delta(|x| - 1)\,. \tag{67}$$

One can show that the Haar integration satisfies

$$\int_{\mathrm{SU}(2)} d[U] U_{ab} U^\dagger_{cd} = \frac{1}{2}\delta_{ad}\delta_{bc}\,. \tag{68}$$

It is straightforward to show this using the explicit parametrization that we just obtained, however, we can also check this against our simple stochastic sampling implementation discussed in the previous subsection.

Stochastic integration using Haar measure

In [3]:

```
import gpt as g
import numpy as np
import scipy.linalg

pauli = [np.matrix([[0,1],[1,0]]), np.matrix([[0,-1j],[1j,0]]), np.matrix([[1,0],[0,-1]])]
U = [np.matrix(scipy.linalg.expm(2j*sum([np.random.normal()*p for p in pauli]))) for i in range(40)]
for u in U:
    assert np.linalg.norm(u * u.H - np.identity(2)) < 1e-14

def haar_integrate(f, U, N, NskipToMakeIndependent):
    u0 = U[np.random.randint(0,len(U))]
    val = 0.0
    val2 = 0.0
    n = 0.0
    for i in range(N):
        v = f(u0).real
        val += v
        val2 += v**2
        n += 1
        for j in range(NskipToMakeIndependent):
            u0 = U[np.random.randint(0,len(U))] * u0
    # just use biased estimator for error
    return (val/n, (val2/n - (val/n)**2)**0.5 / n**0.5 )

for a in range(2):
    for b in range(2):
        for c in range(2):
            for d in range(2):
                print(f"\int dU U_{a}{b} U^\dagger_{c}{d} =",
                      haar_integrate(lambda u: u[a,b]*u.H[c,d], U, 10000, 2))
```

```
\int dU U_00 U^\dagger_00 = (0.4963044822891241, 0.002874731877928811)
\int dU U_00 U^\dagger_01 = (0.007828394800862332, 0.0028740171419067277)
\int dU U_00 U^\dagger_10 = (-0.001844956947146266, 0.0028647142161262475)
\int dU U_00 U^\dagger_11 = (0.0026751189098447098, 0.0040610681617341)
\int dU U_01 U^\dagger_00 = (0.0004471683358340047, 0.0029073253833936242)
\int dU U_01 U^\dagger_01 = (0.0034909122072718794, 0.004106166013781911)
\int dU U_01 U^\dagger_10 = (0.4978861456473543, 0.002893069729484938 5)
\int dU U_01 U^\dagger_11 = (0.0026898438724813297, 0.0028914647379603485)
\int dU U_10 U^\dagger_00 = (0.000477677892407116, 0.002863568675674644)
\int dU U_10 U^\dagger_01 = (0.49465169552991295, 0.0029002791419250867)
\int dU U_10 U^\dagger_10 = (0.0025135099129404356, 0.004021598829467 53)
\int dU U_10 U^\dagger_11 = (0.008233227681888599, 0.0028798633019951943)
\int dU U_11 U^\dagger_00 = (0.004582479741889893, 0.004133579983541516)
\int dU U_11 U^\dagger_01 = (0.0020417697565920964, 0.0028761547175931634)
\int dU U_11 U^\dagger_10 = (-0.004925965656714123, 0.0028903740046162135)
\int dU U_11 U^\dagger_11 = (0.4987628291665141, 0.002880979866428887)
```

Haar measure using invariant length elements

Before moving on to the heat bath, we observe that there is also another straightforward way to determine the Haar integration measure by studying invariant length elements

$$ds^2 = \mathrm{Tr}\left[(dUU^\dagger)(dUU^\dagger)^\dagger\right] \tag{69}$$

that by construction is invariant under $U \to VU$ and $U \to UV$. For the angular parametrization of SU(2), this yields

$$ds^2 = 2d\varphi_0^2 + 2d\varphi_1^2\sin(\varphi_0)^2 + 2d\varphi_2^2\sin(\varphi_0)^2\sin(\varphi_1)^2 = \begin{pmatrix} d\varphi_0 & d\varphi_1 & d\varphi_2 \end{pmatrix} G \begin{pmatrix} d\varphi_0 \\ d\varphi_1 \\ d\varphi_2 \end{pmatrix} \tag{70}$$

with

$$G = \begin{pmatrix} 2 & & \\ & 2\sin(\varphi_0)^2 & \\ & & 2\sin(\varphi_0)^2\sin(\varphi_1)^2 \end{pmatrix}. \tag{71}$$

Therefore there is a different basis of 1-forms $d\beta_\mu$ for which the length element is simply

$$ds^2 = d\beta_0^2 + d\beta_1^2 + d\beta_2^2 \tag{72}$$

corresponding to the integration measure

$$d[U] = d\beta_0 d\beta_1 d\beta_2 \tag{73}$$

and

$$\begin{pmatrix} d\beta_0 \\ d\beta_1 \\ d\beta_2 \end{pmatrix} = G^{1/2}\begin{pmatrix} d\varphi_0 \\ d\varphi_1 \\ d\varphi_2 \end{pmatrix} \tag{74}$$

and thus

$$d[U] = d\varphi_0 d\varphi_1 d\varphi_2 \sqrt{\det(G)} = \sqrt{8} d\varphi_0 d\varphi_1 d\varphi_2 \sin(\varphi_0)^2\sin(\varphi_1) \tag{75}$$

which agrees with our previous result up to the constant factor that only changes the normalization of the integral. Since we could re-define the invariant length-element ds^2 by a constant prefactor, this procedure gives us the properly normalized Haar measure only after normalizing the integral to $\int d[U] = 1$.

Heat bath algorithm

Let us now consider the heat bath algorithm to perform the integral over a single gauge link $U \in$ SU(2). In order to extend this to integrals over SU(3) or more generally SU(N_c), we can then consider the SU(2) subgroups of SU(N_c) and re-write the integral over SU(N_c) as a sufficient number of integrals over SU(2) subgroups. This is what the heat bath studied numerically in the last lecture does.

With $U \in$ SU(2), we need to integrate over

$$dP(U) = d[U]\exp\left(\frac{\beta}{2}\mathrm{Re}\,\mathrm{Tr}\,UA\right) \tag{76}$$

with staple matrix A, which itself is a sum over SU(2) matrices. As we have seen above, each SU(2) matrix can be written of the form $\Sigma \cdot x$ such that sums of SU(2) matrices always have to be proportional to a SU(2) matrix again. Therefore, we can make the ansatz

$$A = aV \tag{77}$$

with

$$a = \sqrt{\det(A)} \,. \tag{78}$$

It can also be shown that $\det(A) \geq 0$. If $\det(A) = 0$, then $a = 0$ and $A = 0$ such that we can simply integrate over the Haar measure. If $a \neq 0$, we proceed as follows.

We define a matrix $X = UV$ in SU(2) and use the Haar invariance $d[U] = d[X]$ to find

$$dP(U) = d[X] \exp\left(\frac{\beta a}{2} \mathrm{Re}\,\mathrm{Tr}\, X\right) . \tag{79}$$

The strategy is now to generate matrices X following this distribution and to use $U = XV^\dagger = XA^\dagger / a$ to recover the desired gauge link U. We write

$$X = \Sigma \cdot x \tag{80}$$

with

$$d[X] = \frac{1}{2\pi^2} dx_0 dx_1 dx_2 dx_3 \delta(|x| - 1) \tag{81}$$

and

$$\mathrm{Tr}\, X = 2x_0 \tag{82}$$

such that

$$dP(U) = \frac{1}{2\pi^2} dx_0 dx_1 dx_2 dx_3 \delta(|x| - 1) \exp(\beta a x_0) \,. \tag{83}$$

We note that only x_0 remains in the exponential and therefore it is useful to separate integral by writing

$$dP(U) = \frac{1}{2\pi^2} dx_0 dx_1 dx_2 dx_3 \delta(\sqrt{x_0^2 + |\vec{x}|^2} - 1) \exp(\beta a x_0) \tag{84}$$

$$= \frac{1}{2\pi^2} dx_0 dx_1 dx_2 dx_3 \frac{\Theta(1 - x_0^2)}{|\vec{x}|} \delta(|\vec{x}| - \sqrt{1 - x_0^2}) \exp(\beta a x_0) \,. \tag{85}$$

with $\vec{x}^T = (x_1, x_2, x_3)$. We now write $\vec{x}$ in spherical coordinates r, φ, θ such that

$$dP(U) = \frac{1}{2\pi^2} dx_0 d\theta d\varphi dr r^2 \sin(\theta) \frac{\Theta(1 - x_0^2)}{r} \delta(r - \sqrt{1 - x_0^2}) \exp(\beta a x_0) \tag{86}$$

$$= \frac{1}{2\pi^2} dx_0 d\theta d\varphi dr \sin(\theta) \Theta(1 - x_0^2) \sqrt{1 - x_0^2} \delta(r - \sqrt{1 - x_0^2}) \exp(\beta a x_0) \,. \tag{87}$$

We now need to sample x_0 from the distribution

$$dp(x_0) = dx_0 \Theta(1 - x_0^2) \sqrt{1 - x_0^2} \exp(\beta a x_0) \tag{88}$$

which can be performed by first defining

$$x_0 = 1 - 2\lambda^2 \tag{89}$$

such that we need to sample $\lambda \in [0, 1]$ from

$$dp(\lambda) \propto d\lambda \lambda^2 \sqrt{1 - \lambda^2} \exp(-2\beta a \lambda^2) \tag{90}$$

this in turn can be generated by sampling four random numbers r_1, r_2, r_3, r_4 uniformly distribute in $]0, 1]$ and defining

$$\lambda^2 = -\frac{1}{2a\beta} (\ln(r_1) + \cos(2\pi r_2)^2 \ln(r_3)) \tag{91}$$

which we accept only if $r_4^2 \leq 1 - \lambda^2$.

The value r is then fixed by $r = \sqrt{1 - x_0^2}$. The remaining random variables $z = \cos(\theta)$ and φ can then be sampled from a uniform distribution $z \in [-1, 1]$ and $\varphi \in [0, 2\pi[$.

The implementation of this algorithm in GPT can be studied here.

The Langevin algorithm

The Langevin algorithm demonstrated above generates a Markov chain by integrating a stochastic differential equation. We will illustrate the algorithm for the simple case of a continuous random variable x following the distribution

$$p(x) = e^{-S(x)} \,, \tag{92}$$

however, the generalization of the algorithm to integrals over gauge groups is straightforward albeit tedious.

We define a discrete stochastic process by

$$x_{n+1} = x_n + \Delta_n(x_n) \tag{93}$$

with

$$\Delta_n(x) = -\varepsilon S'(x) + \sqrt{\varepsilon} \eta_n \tag{94}$$

with η satisfying

$$\langle \eta_n^{2N+1} \rangle = 0 \,, \qquad \langle \eta_n \eta_m \rangle = 2\delta_{nm} \tag{95}$$

for all integer N. The details of distribution of η beyond the first two moments do not matter at this point.

Fokker-Planck equation

For now, let us assume there is a stationary probability distibution of x_n for sufficiently large n. Then the probability $P(n,x)$ to find the value x at step n follows from the **Fokker-Planck** equation

$$\begin{aligned} P(n+1,x_{n+1}) &= \int dx_n P(n,x_n \cap n+1, x_{n+1}) && (96) \\ &= \int dx_n P(n+1,x_{n+1}|n,x_n)P(n,x_n) && (97) \end{aligned}$$

with

$$\begin{aligned} P(n+1,x_{n+1}|n,x_n) &= \langle \delta\left(x_{n+1}-x_n-\Delta_n(x_n)\right)\rangle && (98) \\ &= \delta(x_{n+1}-x_n) \\ &\quad + \varepsilon\left[S'(x_n)\delta'(x_{n+1}-x_n) + \delta''(x_{n+1}-x_n)\right] \\ &\quad + O(\varepsilon^2)\,. && (99) \end{aligned}$$

We can define the operation of a transition matrix T on a function f as

$$\begin{aligned} (T\circ f)(x) &\equiv \int dy P(n+1,x|n,y)f(y) && (100) \\ &= \left\{1+\varepsilon\partial_x\left[S'(x)+\partial_x\right] + O(\varepsilon^2)\right\}f(x)\,. && (101) \end{aligned}$$

Therefore

$$P(n+1,x_{n+1}) = (T\circ P(n))(x_{n+1}) \qquad (102)$$

and more generally

$$P(n+N,x_{n+N}) = (T^N\circ P(n))(x_{n+N}) \qquad (103)$$

with $T^N\circ f = T^{N-1}\circ(T\circ f)$ and $T^1\circ f = f$.

Stationary probability distribution

We now need to study the right-eigensystem of T. A stationary probability distribution $\overline{P}$ requires the presence of an eigenvalue 1 and corresponds to its eigenvector. All other eigenvalues need to be smaller than one such that the stationary distribution is approached in the limit $n\to\infty$.

The stationary distribution exists if $T\overline{P}=\overline{P}$ or equivalently

$$0 = \partial_x\left[\overline{P}S' + \overline{P}'\right] + O(\varepsilon) \qquad (104)$$

or

$$\overline{P}'(x) = -\overline{P}(x)S'(x) + c_1 + O(\varepsilon) \qquad (105)$$

has a solution for a constant c_1. This is solved by

$$\overline{P}(x) = c_0 e^{-S(x)} + c_1 e^{-S(x)}\int_1^x dy\, e^{S(y)} \qquad (106)$$

for all c_0. If we now insist that $\overline{P}$ and its derivative vanish for sufficiently large x, only $c_1=0$ is allowed. On compact spaces one similarly imposes the compactification on $\overline{P}$ and its derivatives to determine additional integration constants. The overall constant is determined by probability conservation.

Therefore we have shown that if a stationary probability distribution exists, it follows the desired

$$\overline{P}(x) = e^{-S(x)}\,. \qquad (107)$$

We next show that the system indeed converges to a stationary probability distribution up to the $O(\varepsilon)$ corrections which we have previously neglected. One can construct higher-order stochastic differential equations, which eliminate additional powers of ε.

Convergence

A similarity transform of the operator

$$\Delta T = \partial_x\left[S'(x)+\partial_x\right] \qquad (108)$$

by

$$\begin{aligned} \tilde{\Delta}T &= e^{S(x)/2}\Delta T e^{-S(x)/2} && (109) \\ &= e^{S(x)/2}\partial_x e^{-S(x)/2}\left[\frac{1}{2}S'(x)+\partial_x\right] && (110) \\ &= -\left[\frac{1}{2}S'(x)-\partial_x\right]\left[\frac{1}{2}S'(x)+\partial_x\right] && (111) \\ &= -Q^\dagger Q && (112) \end{aligned}$$

with

$$Q = \frac{1}{2}S'(x)+\partial_x \qquad (113)$$

shows that the self-adjoint operator $\tilde{\Delta}T$ is real and semi-negative definite. Therefore, the stochastic process is driven towards the stationary solution $\overline{P}$.

The implementation of this algorithm in GPT can be studied here.

Homework: Calculate $\langle x^2 \rangle$ for x following a normal distribution with 0 mean and variance 1 by implementing a Langevin process.

```
In [ ]:
```