Chapter 11: symmetries of the Wilson action

In this chapter we study in more detail the symmetries of the lattice action of QCD. In order to do so, we first remind ourselves of the full lattice action with gauge links U_μ and a single fermion field ψ. We have

$$S = \sum_x \bar\psi(x) \left(\sum_\mu \gamma_\mu D_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{D}_\mu \overrightarrow{D}_\mu \right) \psi(x) + \beta \sum_x \sum_{\mu<\nu} (1 - P_{\mu\nu}(x)) \tag{1}$$

with

$$\begin{aligned} D_\mu &= \frac{1}{2}\left(\overleftarrow{D}_\mu + \overrightarrow{D}_\mu\right) = \frac{1}{2}\left(C_\mu^+ - C_\mu^-\right) \,, && (2)\\ \overleftarrow{D}_\mu \overrightarrow{D}_\mu &= C_\mu^+ + C_\mu^- - 2\,, && (3)\\ C_\mu^+ \psi(x) &= U_\mu(x)\psi(x+a\hat\mu)\,, && (4)\\ C_\mu^- \psi(x) &= U_\mu^\dagger(x-a\hat\mu)\psi(x-a\hat\mu)\,, && (5)\\ P_{\mu\nu}(x) &= \frac{1}{N_c}\mathrm{Re}\,\mathrm{Tr}\,U_{\mu\nu}(x)\,, && (6)\\ U_{\mu\nu}(x) &= U_\mu(x)U_\nu(x+a\hat\mu)U_\mu(x+a\hat\nu)^\dagger U_\nu(x)^\dagger\,. && (7) \end{aligned}$$

Particle and antiparticle solutions in Minkowski space

In chapter 5 we learned that the wavefunction of a free Dirac particle in Minkowski space satisfies

$$(i\gamma^0\partial_0 - m)\psi(t) = 0 \tag{8}$$

if we assume that it is invariant in space, i.e., $\partial_i\psi = 0$. This equation has solutions

$$\psi^+(t) = (1-\gamma^0)e^{imt}\psi_0^+ \tag{9}$$

and

$$\psi^-(t) = (1+\gamma^0)e^{-imt}\psi_0^- \tag{10}$$

since $\gamma^0(1-\gamma^0) = -(1-\gamma^0)$ and $\gamma^0(1+\gamma^0) = 1+\gamma^0$. Here $\psi_0^\pm$ are vectors selecting a spin configuration. We can think of ψ^+ as the **particle** and ψ^- as the **antiparticle** solution.

We can now consider the **charge conjugation** transformation C that maps particles to antiparticles and vice versa. Since $(\gamma^0)^* = \gamma^0$ and

$$i\gamma^2(1\pm\gamma^0)i\gamma^2 = 1\mp\gamma^0$$

we find

$$\begin{aligned} C\psi^\pm(t) &\equiv -i\gamma^2(\psi^\pm(t))^* = -i\gamma^2(1\mp\gamma^0)e^{\mp imt}(\psi_0^\pm)^* && (11)\\ &= i\gamma^2(1\mp\gamma^0)i\gamma^2 e^{\mp imt} i\gamma^2(\psi_0^\pm)^* && (12)\\ &= (1\pm\gamma^0)e^{\mp imt} i\gamma^2(\psi_0^\pm)^* && (13)\\ &= \psi^\mp(t) && (14) \end{aligned}$$

if we identify

$$i\gamma^2(\psi_0^\pm)^* = \psi_0^\mp\,. \tag{15}$$

The charge conjugation operation defined in this way therefore relates particle solutions of a given spin to antiparticle solutions of a different spin configuration.

We can generalize this discussion beyond free-field solutions studying the charge conjugation operation C that replaces

$$\begin{aligned} \psi \to \psi^C &= -i\gamma^2\psi^* = -i\gamma^2(\psi^\dagger)^T = -i\gamma^2(\bar\psi\gamma^0)^T && (16)\\ &= -i\gamma^2\gamma^0\bar\psi^T\,. && (17) \end{aligned}$$

Charge-conjugation symmetry

In Euclidean space this transformation can be written as

$$\psi \to \psi^C = C^{-1}\bar\psi^T\,, \tag{18}$$

where $C^{-1} = \gamma^2\gamma^4$ with Euclidean gamma matrices. Note that it is somewhat more convenient to define instead

$$C^{-1} = i\gamma^2\gamma^4 \tag{19}$$

by including a global phase factor such that $C^{-1} = C = C^\dagger = -C^T$. It is straightforward to show that

$$C\gamma^\mu C = -(\gamma^\mu)^T \tag{20}$$

which for the free theory again yields a symmetry of the action under

$$\psi \to C\bar\psi^T \tag{21}$$

such that

$$\begin{aligned}\bar\psi &\to (C\bar\psi^T)^\dagger\gamma^4 = (\bar\psi^T)^\dagger C\gamma^4 && (22)\\ &= -(\bar\psi^T)^\dagger(\gamma^4)^T C = -(\gamma^4\bar\psi^\dagger)^T C && (23)\\ &= -\psi^T C\,. && (24)\end{aligned}$$

The free action therefore transforms as

$$\begin{aligned}S &= \sum_x \bar\psi(x)\left(\sum_\mu \gamma_\mu\partial_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{\partial}_\mu\overrightarrow{\partial}_\mu\right)\psi(x) && (25)\\ \to S^C &\equiv -\sum_x \psi(x)^T C\left(\sum_\mu \gamma_\mu\partial_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{\partial}_\mu\overrightarrow{\partial}_\mu\right)C\bar\psi(x)^T && (26)\\ &= -\sum_x \psi(x)^T\left(\sum_\mu(-\gamma_\mu)^T\partial_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{\partial}_\mu\overrightarrow{\partial}_\mu\right)\bar\psi(x)^T\,. && (27)\end{aligned}$$

Since ψ and $\bar\psi$ are Grassmannian variables, they anticommute. In addition for commuting variables

$$\sum_x f(x)g(x+\hat\mu) = \sum_x g(x)f(x-\hat\mu) \tag{28}$$

such that the first derivative term gets an additional minus sign. Finally $v^T X^T w^T = wXv$. In total, we find

$$S^C = \sum_x \bar\psi(x)\left(\sum_\mu \gamma_\mu\partial_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{\partial}_\mu\overrightarrow{\partial}_\mu\right)\psi(x) = S\,. \tag{29}$$

The free Wilson fermion action is therefore invariant under charge conjugation.

In order to preserve a symmetry in the interacting case, we need the covariant shifts $C^\pm$ to transform as

$$\sum_x f(x)C^\pm g(x) \to \sum_x g(x)^T C^\mp f(x)^T\,. \tag{30}$$

Concretely the transformed gauge fields

$$U_\mu \to U_\mu^C$$

in $C^\pm$ need to obey

$$\sum_x f(x)U_\mu^C(x)g(x+a\hat\mu) = \sum_x g(x)^T U_\mu^\dagger(x-a\hat\mu)f(x-a\hat\mu)^T = \sum_x g(x+a\hat\mu)^T U_\mu^\dagger(x)f(x)^T = \sum_x f(x)U_\mu^*(x)g(x+a\hat\mu)\,. \tag{31}$$

This is satisfied if we demand that the gauge fields transform under charge conjugation symmetry as

$$U_\mu \to U_\mu^C = U_\mu^*\,. \tag{32}$$

Finally, we need to check if the gauge-field part of the action also preserves this transformation. We note that under $U_\mu \to U_\mu^* = (U_\mu^T)^\dagger$ we have

$$\begin{aligned}U_{\mu\nu}(x) &\to (U_\mu^\dagger(x))^T(U_\nu^\dagger(x+a\hat\mu))^T U_\mu(x+a\hat\nu)^T U_\nu(x)^T && (33)\\ &= (U_\nu(x)U_\mu(x+a\hat\nu)(U_\nu^\dagger(x+a\hat\mu))(U_\mu^\dagger(x)))^T && (34)\\ &= U_{\nu\mu}(x)^T && (35)\end{aligned}$$

such that

$$\mathrm{Tr}\,U_{\mu\nu}(x) \to \mathrm{Tr}\,U_{\nu\mu}(x)\,. \tag{36}$$

Finally, we note that $U_{\mu\nu}(x)^\dagger = U_{\nu\mu}(x)$ such that due to the real part $P_{\mu\nu}$ and therefore also the full Wilson action is invariant under charge conjugation symmetry.

γ_5-Hermiticity

Next, we prove the γ_5-Hermiticity (see chapter 10) of the Wilson Dirac operator. The Dirac operator is given by

$$D(U) = \sum_\mu \gamma_\mu D_\mu + m - \frac{1}{2}\sum_\mu \overleftarrow{D}_\mu\overrightarrow{D}_\mu\,. \tag{37}$$

We first note that

$$(C_\mu^+)^\dagger = (U_\mu S_\mu^+)^\dagger = (S_\mu^+)^\dagger U_\mu^\dagger = S_\mu^- U_\mu^\dagger = C_\mu^- \tag{38}$$

since for any f and g

$$\begin{aligned}\sum_x f(x)^\dagger(S_\mu^+)^\dagger g(x) &= \sum_x (g(x)^\dagger S_\mu^+ f(x))^\dagger = \sum_x (g(x)^\dagger f(x+a\hat\mu))^\dagger = \sum_x f(x+a\hat\mu)^\dagger g(x) && (39)\\ &= \sum_x f(x)^\dagger g(x-a\hat\mu) = \sum f(x)^\dagger S_\mu^- g(x) && (40)\end{aligned}$$

and therefore $(S_\mu^+)^\dagger = S_\mu^-$.

Equipped with these identities, we find that

$$
\begin{aligned}
\gamma_5 D(U)^\dagger \gamma_5 &= \gamma_5 \left(\sum_\mu \gamma_\mu (D_\mu)^\dagger + m - \frac{1}{2} \sum_\mu (\overleftarrow{D}_\mu \overrightarrow{D}_\mu)^\dagger \right) \gamma_5 && (41)\\
&= \left(-\sum_\mu \gamma_\mu (D_\mu)^\dagger + m - \frac{1}{2} \sum_\mu (\overleftarrow{D}_\mu \overrightarrow{D}_\mu)^\dagger \right) && (42)\\
&= \left(\sum_\mu \gamma_\mu D_\mu + m - \frac{1}{2} \sum_\mu \overleftarrow{D}_\mu \overrightarrow{D}_\mu \right) && (43)\\
&= D(U)\,. && (44)
\end{aligned}
$$

This is equivalent to

$$
(\gamma_5 D(U))^\dagger = \gamma_5 D(U) \tag{45}
$$

and

$$
(D(U)\gamma_5)^\dagger = D(U)\gamma_5\,. \tag{46}
$$

Space-time symmetries

Let us consider an n-dimensional lattice $\Lambda = \{\vec{n} \in \mathbb{Z}^n\}$ and operations $g : \Lambda \to \Lambda$ that can be expressed as a sequence of axis exchange operations $A_{\mu\nu}$ and axis reflection operations R_μ with

$$
\begin{aligned}
(A_{\mu\nu}\vec{n})_\rho &= \begin{cases} \vec{n}_\rho & \text{if } \rho \neq \mu \wedge \rho \neq \nu\,, \\ \vec{n}_\mu & \text{if } \rho = \nu\,, \\ \vec{n}_\nu & \text{if } \rho = \mu\,, \end{cases}\,, && (47)\\
(R_\mu \vec{n})_\rho &= \begin{cases} \vec{n}_\rho & \text{if } \rho \neq \mu\,, \\ -\vec{n}_\rho & \text{if } \rho = \mu\,, \end{cases}\,, && (48)
\end{aligned}
$$

where it is sufficient to consider $A_{\mu\nu}$ with $\mu < \nu$.

Rotations, e.g., can be written as a combination of both. Take the rotation along the z axis of a three-dimensional vector

$$
\begin{pmatrix} x \\ y \\ z \end{pmatrix} \to \begin{pmatrix} y \\ -x \\ z \end{pmatrix} \tag{49}
$$

which can be written as a $R_1 \circ A_{01}$.

In n dimensions we therefore have

$$
n! 2^n \tag{50}
$$

unique non-translation space-time symmetries on a lattice. If we consider the symmetries of the Hamiltonian, $n = 3$ is relevant and we find 48 unique operations. The corresponding group is called the full cubic group O_h and the study of its irreducible representations yields good quantum numbers of the Hamiltonian. We will return to this later.

For the action $n = 4$ is relevant and we have 384 unique operations. Since all of them can be written as combinations of axis-exchange and reflection operations it is sufficient to study the behavior of the action under this restricted set of operations. The corresponding group is called hypercubic group.

Under exchange of axis, we re-label the Lorentz index of the gluon fields U_μ accordingly and apply $A_{\mu\nu}$ to the coordinates x. The symmetry of the gluon Wilson action under axis exchange symmetry is trivial to see. Finally, we need to consider transformations of the fermion fields. We find axis exchange symmetry of the entire action if we transform ψ and $\bar{\psi}$ such that

$$
\bar{\psi}\gamma_\rho\psi = \bar{\psi}\gamma_{\rho'}\psi \tag{51}
$$

where $\rho' = \rho$ if $\mu \neq \rho \wedge \nu \neq \rho$ and $\rho' = \mu$ if $\rho = \nu$ and $\rho' = \nu$ if $\rho = \mu$. It can be shown that using a change of basis in spinor space for ψ and $\bar{\psi}$ such transformations can always be found.

The above symmetries also apply to a lattice with finite number of spatial sites N_μ in each direction. For the reflection symmetry it is convenient to consider lattice sites to run from $-N_\mu/2 + 1$ to $N_\mu/2$. In this convention, the action also exhibits a reflection symmetry R_μ. It is, however, more convenient to study instead the set of reflections of all orthogonal directions to μ that we write as P_μ. As an example in four dimensions $P_0 = R_1 R_2 R_3$ which can be inverted to $R_0 = P_1 P_2 P_3$. The action can be shown to be invariant under all P_μ if

$$
\begin{aligned}
\psi(x) &\to \gamma_\mu \psi(P_\mu x)\,, && (52)\\
\bar{\psi}(x) &\to \bar{\psi}(P_\mu x)\gamma_\mu\,, && (53)\\
U_\mu(x) &\to U_\mu(P_\mu x)\,, && (54)\\
U_\nu(x) &\to U_\nu(P_\mu x - a\hat{\nu})^\dagger && (55)
\end{aligned}
$$

for $\nu \neq \mu$. The corresponding calculation is straightforward.

Study of the full cubic group O_h - implications for the eigenstates of the Hamiltonian

In the following, we will study the finite group O_h in more detail in an attempt to understand the resulting good quantum numbers. We will introduce group-theoretic concepts as needed.

As a reminder: A **group** is the tuple $(G, \circ)$ with set G and $a \circ b : G \times G \to G$ such that

- $(\forall a, b, c \in G)((a \circ b) \circ c = a \circ (b \circ c))$
- $(\exists e \in G)(\forall a \in G)(e \circ a = a \circ e = a)$; e is unique
- $(\forall a \in G)(\exists a^{-1} \in G)(a \circ a^{-1} = a^{-1} \circ a = e)$

A group is called **finite** if $|G| \in \mathbb{N}$.

For finite groups these studies can be performed numerically in a straightforward manner. To do so, we first need to construct a set of all group elements that are generated by axis interchange and reflection. We do so by considering the action on a complete basis of Λ for $n = 3$ and demonstrate that the set is closed under the generating operations.

Our first task is to associate for each $g_i \in G$ an index $i \in [0, \ldots, |G| - 1]$ and to create a **multiplication table** $M \in [0, \ldots, |G| - 1]^{|G| \times |G|}$ with elements

$$M_{ij} = l \quad \leftrightarrow \quad g_j \circ g_i = g_l \,.$$

To simplify the discussion, we first study the group O that is made up of all rotations before studying the full cubic group O_h.

In [1]:
```
import numpy as np

basis = [[1,0,0],[0,1,0],[0,0,1]]

def interchange(x,a,b):
    xp=[xi for xi in x]
    xp[a]=x[b]
    xp[b]=x[a]
    return xp

def reflection(x,a):
    xp=[xi for xi in x]
    xp[a]=-x[a]
    return xp

def rotateZ90(x):
    return [-x[1],x[0],x[2]]

def rotateX90(x):
    return [x[0],-x[2],x[1]]

def find_closure(actions):
    elements = [basis]
    sequence = { str(basis) : [] }
    n = 0
    while n != len(elements):
        n = len(elements)
        for a in actions:
            for e in elements:
                ep = a(e)
                if ep not in elements:
                    elements.append(ep)
                    sequence[str(ep)] = sequence[str(e)] + [a]

    return elements, sequence

# First only study group of rotations, i.e., cubic group O
elements = find_closure([
    #lambda e: [interchange(b,0,1) for b in e],
    #lambda e: [interchange(b,0,2) for b in e],
    #lambda e: [interchange(b,1,2) for b in e],
    #lambda e: [reflection(b,0) for b in e],
    #lambda e: [reflection(b,1) for b in e],
    #lambda e: [reflection(b,2) for b in e],
    lambda e: [rotateZ90(b) for b in e],
    lambda e: [rotateX90(b) for b in e]
])

print(len(elements[0]))
```

```
24
```

In [2]:
```
def apply(el, e0, idx):
    for acc in el[1][str(el[0][idx])]:
        e0 = acc(e0)
    return e0

def multiplication_table(el, ba):
    n=len(el[0])
    M=[[-1 for i in range(n)] for j in range(n)]
    for i in range(n):
        ei = apply(el, ba, i)
        for j in range(n):
            eji = apply(el, ei, j)
            M[i][j] = el[0].index(eji)
    return M


mt = multiplication_table(elements, basis)

print(np.array(mt))
```

```
[[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23]
 [ 1  2  3  0  5  6  7  4  9 10 11  8 13 14 15 12 17 18 19 16 21 22 23 20]
 [ 2  3  0  1  6  7  4  5 10 11  8  9 14 15 12 13 18 19 16 17 22 23 20 21]
 [ 3  0  1  2  7  4  5  6 11  8  9 10 15 12 13 14 19 16 17 18 23 20 21 22]
 [ 4 16 14 22  8 17  2 21 12 18  6 20  0 19 10 23 11 15  3  7  1 13  9  5]
 [ 5 17 15 23  9 18  3 22 13 19  7 21  1 16 11 20  8 12  0  4  2 14 10  6]
 [ 6 18 12 20 10 19  0 23 14 16  4 22  2 17  8 21  9 13  1  5  3 15 11  7]
 [ 7 19 13 21 11 16  1 20 15 17  5 23  3 18  9 22 10 14  2  6  0 12  8  4]
 [ 8 11 10  9 12 15 14 13  0  3  2  1  4  7  6  5 20 23 22 21 16 19 18 17]
 [ 9  8 11 10 13 12 15 14  1  0  3  2  5  4  7  6 21 20 23 22 17 16 19 18]
 [10  9  8 11 14 13 12 15  2  1  0  3  6  5  4  7 22 21 20 23 18 17 16 19]
 [11 10  9  8 15 14 13 12  3  2  1  0  7  6  5  4 23 22 21 20 19 18 17 16]
 [12 20  6 18  0 23 10 19  4 22 14 16  8 21  2 17  1  5  9 13 11  7  3 15]
 [13 21  7 19  1 20 11 16  5 23 15 17  9 22  3 18  2  6 10 14  8  4  0 12]
 [14 22  4 16  2 21  8 17  6 20 12 18 10 23  0 19  3  7 11 15  9  5  1 13]
 [15 23  5 17  3 22  9 18  7 21 13 19 11 20  1 16  0  4  8 12 10  6  2 14]
 [16 14 22  4 17  2 21  8 18  6 20 12 19 10 23  0 15  3  7 11 13  9  5  1]
 [17 15 23  5 18  3 22  9 19  7 21 13 16 11 20  1 12  0  4  8 14 10  6  2]
 [18 12 20  6 19  0 23 10 16  4 22 14 17  8 21  2 13  1  5  9 15 11  7  3]
 [19 13 21  7 16  1 20 11 17  5 23 15 18  9 22  3 14  2  6 10 12  8  4  0]
 [20  6 18 12 23 10 19  0 22 14 16  4 21  2 17  8  5  9 13  1  7  3 15 11]
 [21  7 19 13 20 11 16  1 23 15 17  5 22  3 18  9  6 10 14  2  4  0 12  8]
 [22  4 16 14 21  8 17  2 20 12 18  6 23  0 19 10  7 11 15  3  5  1 13  9]
 [23  5 17 15 22  9 18  3 21 13 19  7 20  1 16 11  4  8 12  0  6  2 14 10]]
```

In [3]:

```
def find_identity(el, ba):
    return el[0].index(ba)

ident = find_identity(elements, basis)
assert ident == 0 # for convenience always have first element be the identity

print(ident)

def inverse_list(mt, e):
    return [x.index(e) for x in mt]

inv = inverse_list(mt, ident)

print(inv)
```

```
0
[0, 3, 2, 1, 12, 18, 6, 20, 8, 9, 10, 11, 4, 22, 14, 16, 15, 17, 5, 23, 7, 21, 13, 19]
```

In [4]:

```
# test inverses
print([mt[i][inv[i]] for i in range(len(inv))])
print([mt[inv[i]][i] for i in range(len(inv))])
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

It is convenient to identify the index 0 with the identity element e.

Next, we consider $\Gamma_r : G \to \mathrm{GL}(V)$ that maps group elements to invertible matrices acting on a finite-dimensional vector space V. We call Γ_r a **representation** of the group if it preserves its structure, i.e.,

$$(\forall a, b \in G)(\Gamma_r(a)\Gamma_r(b) = \Gamma_r(a \circ b))\,.$$

In other words, it is a group homomorphism between G and the general linear group over vector space V.

If there is a lower-dimensional subspace V' of V for which $(\forall g \in G, v \in V')(\Gamma_r(g)v \in V')$ we call the representation Γ_r **reducible** and the projection of Γ_r to V' a **sub-representation**. In other words Γ_r has a block-matrix form in the appropriate basis. If no such sub-representation exists, we call Γ_r **irreducible**.

If G describes a symmetry of a Hamiltonian H, the Hamiltonian has a corresponding block structure and the spectrum of H can be classified following the irreducible representations. This will be one of the main uses of representation theory in the remainder of this lecture.

So the next question is, given our group G, how many irreducible representations are there? For a finite group there will always be a finite number and there is a straightforward method to find out how many there are.

To this end, we first define a **conjugacy class** C as follows: For all $a, b \in G$ we say a and b are **conjugate** ($a \sim b$) if there is a group element $g \in G$ such that $gag^{-1} = b$. This defines an equivalence relation, i.e., if $a \sim b$ and $b \sim c$ then $a \sim c$. Therefore it splits the group G in separate conjugacy classes, where a member of each class is conjguate to all others but not conjugate to members of a different class.

It is straightforward to construct the conjugacy classes of a finite group, as we will do below, and one can show that the number of conjugacy classes is exactly the number of irreducible representations (irreps) of the group!

In [5]:

```
# find conjugation classes / invariant subspaces
def find_conjugation_classes(mt, inv, ident):
    n=len(inv)
    tostudy=list(range(n))
    classes = []
    while len(tostudy) > 0:
        x = tostudy[0]
        conjugate_to = []
        for i in range(n):
            conjugate_to.append(mt[mt[i][x]][inv[i]])
        conjugate_to = set(conjugate_to)
        for y in conjugate_to:
            tostudy.remove(y)
        classes.append(conjugate_to)
    return classes
```

In [6]:

```
classes = find_conjugation_classes(mt, inv, ident)
print(classes)
```

```
[{0}, {1, 3, 4, 12, 19, 23}, {8, 2, 10}, {5, 7, 13, 15, 16, 18, 20, 22}, {6, 9, 11, 14, 17, 21}]
```

So there will be five irreps for O. Next, we define a few obvious representations and test them.

In [7]:

```
# find a matrix representation of individual elements
def matrix_vector(l):
    # transform a lattice vector
    return np.matrix(elements[0][l]).T

def matrix_trivial(l):
    return np.matrix([[1]])
```

```
def matrix_determinant(l):
    return np.matrix([[np.linalg.det(matrix_vector(l))]], dtype=np.int32)

def matrix_regular(l):
    M = np.zeros(shape=(len(mt),len(mt)), dtype=np.int32)
    for i in range(len(mt)):
        M[i,mt[l][i]] = 1
    return np.matrix(M)
```

In [8]:

```
# test if matrix is a group representation
def test_representation(matrix):
    print(matrix.__name__)
    for i in range(len(mt)):
        for j in range(len(mt)):
            eps = np.linalg.norm(matrix(j) * matrix(i) - matrix(mt[i][j]))
            assert eps < 1e-13

test_representation(matrix_vector)
test_representation(matrix_regular)
test_representation(matrix_trivial)
```

```
matrix_vector
matrix_regular
matrix_trivial
```

Next, we need a test if a representation is irreducible. To this end, we first define the **character** of a representation r

$$\chi_r(g) = \mathrm{Tr}\,\Gamma_r(g)\,.$$

It is straightforward to see that the character of two elements $a, b \in G$ is identical if they belong to the same conjugacy class. We call such functions **class functions**.

Let C be the set of all conjugacy classes of group G, then the matrix

$$(\chi_r)_c$$

for $c \in C$ and irreps r is the so-called **character table** of the group. One writes c in the columns and r in the rows.

Next, we define the inner product of two class functions $\chi : G \to \mathbb{R}$ and $\phi : G \to \mathbb{R}$ as

$$\langle \chi, \phi \rangle = \frac{1}{|G|} \sum_{g \in G} \chi(g)\phi(g)\,.$$

Finally, one can show that $\langle \chi_r, \chi_r \rangle = 1$ if and only if r is irreducible.

We therefore now know how many irreducible representations there are and how to check if a representation is irreducible. We next check the representations that we already constructed above and then figure out a way to construct the remaining irreps.

Before doing so, we note that from Schur's orthogonality theorem it follows that two rows in the character table are orthonormal with respect to the above inner product. We also note that $\chi_r(e)$ is the dimensionality of the irreducible representation. By convention one writes the conjugacy class containing e in the first column of the character table and the trivial representation that maps everything to 1 to the first row.

In [9]:

```
def fmt(row):
    return ", ".join([ "%3.0f" % i for i in row ])

def character_table_row(irrep):
    return [np.trace(irrep(list(c)[0])) for c in classes]

def inner_product(a, b):
    return sum([ai*bi*len(ci) for ai, bi, ci in zip(a, b, classes)]) / len(mt)

characters_trivial = character_table_row(matrix_trivial)
characters_vector = character_table_row(matrix_vector)

print(inner_product(characters_trivial, characters_vector))
print(inner_product(characters_trivial, characters_trivial)) # == 1 -> irreducible
print(inner_product(characters_vector, characters_vector)) # == 1 -> irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
```

```
0.0
1.0
1.0
|C|   1,   6,   3,   8,   6
------------------------------
A1    1,   1,   1,   1,   1
T1    3,   1,  -1,   0,  -1
```

In [10]:

```
characters_regular = character_table_row(matrix_regular)
print(inner_product(characters_regular, characters_regular)) # not irreducible
```

```
24.0
```

The names A1 for the trivial representation and T1 for the representation transforming as a vector are convention but the rows in the character table are the unique fingerprint that defines them.

In order to find all irreps, we can study tensor products of existing representations.

In [11]:

```
# product representation
def product_matrix(A, B):
    def _mat(i):
        return np.kron(A(i),B(i))
    _mat.__name__ = A.__name__ + "_X_" + B.__name__
    return _mat

matrix_vector_vector = product_matrix(matrix_vector, matrix_vector)
test_representation(matrix_vector_vector)
characters_vector_vector = character_table_row(matrix_vector_vector)
print(inner_product(characters_vector_vector, characters_vector_vector)) # not irreducible

#tA=[[1,2],[3,4]]
#tB=[[1.4,2.2],[1.2,3.9]]
#print(np.kron(np.matrix(tA),np.matrix(tB))-np.matrix([[tA[i1][j1]*tB[i2][j2] for j1 in range(2) for j2 in range(2)] for i1 in range(2) for i2 i
```

```
matrix_vector_X_matrix_vector
4.0
```

In [12]:

```
print(inner_product(characters_vector_vector, characters_vector)) # contains both T1 and A1
print(inner_product(characters_vector_vector, characters_trivial))
```

```
1.0
1.0
```

In order to find irreducible representations again, we need to study projections of the product space down to invariant subspaces. One can show that given a representation $\Gamma_r : G \to \mathrm{GL}(V)$ and a character χ_s of an irrep s,

$$P(\chi_s) = \frac{\dim(V)}{|G|} \sum_{g \in G} \Gamma_r(g) \chi_s(g) \tag{56}$$

projects to the corresponding invariant subspace V' of the irreducible representation. Therefore

$$P(\chi_s) \Gamma_r P(\chi_s) \tag{57}$$

is either zero or the irreducible representation s.

In [13]:

```
def projection(matrix, character_row):
    ret = 0.0 * matrix(ident)
    for chi, c in zip(character_row, classes):
        for ci in c:
            ret += matrix(ci)*chi
    P = np.matrix(ret * character_row[0] / len(mt))
    assert np.linalg.norm(P*P - P) < 1e-14
    return P

print(projection(matrix_vector, characters_vector))
print(projection(matrix_vector, characters_trivial))

print(projection(matrix_trivial, characters_vector))
print(projection(matrix_trivial, characters_trivial))
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
[[0.]]
[[1.]]
```

In [14]:

```
print(projection(matrix_vector_vector, characters_vector))
print(projection(matrix_vector_vector, characters_trivial))
```

```
[[ 0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.5  0.  -0.5  0.   0.   0.   0.   0. ]
 [ 0.   0.   0.5  0.   0.   0.  -0.5  0.   0. ]
 [ 0.  -0.5  0.   0.5  0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0. ]
 [ 0.   0.   0.   0.   0.   0.5  0.  -0.5  0. ]
 [ 0.   0.  -0.5  0.   0.   0.   0.5  0.   0. ]
 [ 0.   0.   0.   0.   0.  -0.5  0.   0.5  0. ]
 [ 0.   0.   0.   0.   0.   0.   0.   0.   0. ]]
[[0.33333333 0.         0.         0.         0.33333333 0.
  0.         0.         0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.33333333 0.         0.         0.         0.33333333 0.
  0.         0.         0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.        ]
 [0.33333333 0.         0.         0.         0.33333333 0.
  0.         0.         0.33333333]]
```

It is straightforward to show that properties such as symmetrization/antisymmetrization of basis vectors in the product space create invariant subspaces. These subspaces may or may not correspond to irreducible representations, however, they already reduce the dimensionality of the relevant space and therefore are powerful tools to identify the remaining irreps.

In [15]:

```
def antisymmetric_projector(n):
    ei = [np.zeros(shape=(n,)) for i in range(n)]
    for i in range(n):
        ei[i][i] = 1

    # 00 01 10 11
    # (|01> - |10>)(<01| - <10|) / 2
    # v_i v_j -> v_i v_j - v_j v_i
    P = sum([
        np.matrix(np.outer(np.kron(ei[i],ei[j]) - np.kron(ei[j],ei[i]),
                           np.kron(ei[i],ei[j]) - np.kron(ei[j],ei[i])))/2
     for i in range(n) for j in range(i)])

    assert np.linalg.norm(P*P - P) < 1e-14
    return P

def symmetric_projector(n):
    ei = [np.zeros(shape=(n,)) for i in range(n)]
    for i in range(n):
        ei[i][i] = 1

    # 00 01 10 11
    # (|01> + |10>)(<01| + <10|) / 2 + |00><00| + |11><11|
    # v_i v_j -> v_i v_j + v_j v_i
    P = sum([
        np.matrix(np.outer(np.kron(ei[i],ei[j]) + np.kron(ei[j],ei[i]),
                           np.kron(ei[i],ei[j]) + np.kron(ei[j],ei[i])))/4.
     for i in range(n) for j in range(n)])

    assert np.linalg.norm(P*P - P) < 1e-14
    return P

def diagonal_projector(n):
    ei = [np.zeros(shape=(n,)) for i in range(n)]
    for i in range(n):
        ei[i][i] = 1

    # |00><00| + |11><11|
    # v_i v_j -> v_i v_j \delta_{ij}
    P = sum([
        np.matrix(np.outer(np.kron(ei[i],ei[i]),
                           np.kron(ei[i],ei[i])))
     for i in range(n)])

    assert np.linalg.norm(P*P - P) < 1e-14
    return P

def invert_projector(projector):
    def _proj(n):
        P = projector(n)
        P = np.matrix(np.eye(n*n)) - P
        assert np.linalg.norm(P*P - P) < 1e-14
        return P
    return _proj

def traceless_projector(n):
    ei = [np.zeros(shape=(n,)) for i in range(n)]
    for i in range(n):
        ei[i][i] = 1

    # (|ii>-\sum_l |ll> / ndim)(<ii|-\sum_l <ll| / ndim) + sum_{i!=j} |ij><ij|
    # v_i v_j -> v_i v_j - v_l v_l \delta_{ij} / ndim
    P = sum([
        np.matrix(np.outer(np.kron(ei[i],ei[j]),np.kron(ei[i],ei[j])))
     for i in range(n) for j in range(n) if i != j])

    sll = sum([np.kron(ei[l],ei[l]) for l in range(n)]) / n

    P += sum([
        np.matrix(np.outer(np.kron(ei[i],ei[i]) - sll,np.kron(ei[i],ei[i]) - sll))
     for i in range(n)])

    assert np.linalg.norm(P*P - P) < 1e-14
    return P

# product representation
def projected_product_matrix(A, B, projectors):

    n1 = len(A(0))
    n2 = len(B(0))
    assert n1 == n2

    P = projectors[0](n1)
    for p in projectors[1:]:
        P = P * p(n1)

    def _mat(i):
        return P*np.kron(A(i),B(i))*P
    _mat.__name__ = A.__name__ + "_projected_" + B.__name__
    return _mat
```

In [16]:

```
matrix_a_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                         [antisymmetric_projector])
test_representation(matrix_a_vector_vector)
characters_a_vector_vector = character_table_row(matrix_a_vector_vector)
print(inner_product(characters_a_vector_vector, characters_a_vector_vector)) # is irreducible


print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
print("AVV",fmt(characters_a_vector_vector)) # but unfortunately already known, it is the T1 representation!
```

```
matrix_vector_projected_matrix_vector
1.0
|C|   1,   6,   3,   8,   6
------------------------------
A1    1,   1,   1,   1,   1
T1    3,   1,  -1,   0,  -1
AVV   3,   1,  -1,   0,  -1
```

In [17]:
```
matrix_s_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                  [symmetric_projector])
test_representation(matrix_s_vector_vector)
characters_s_vector_vector = character_table_row(matrix_s_vector_vector)
print(inner_product(characters_s_vector_vector, characters_s_vector_vector)) # reducible
```

```
matrix_vector_projected_matrix_vector
3.0
```

In [18]:
```
# only look at off-diagonal part
matrix_s_offdiag_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                  [invert_projector(diagonal_projector),symmetric_projector])
test_representation(matrix_s_offdiag_vector_vector)
characters_s_offdiag_vector_vector = character_table_row(matrix_s_offdiag_vector_vector)
print(inner_product(characters_s_offdiag_vector_vector, characters_s_offdiag_vector_vector)) # irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
print("T2 ",fmt(characters_s_offdiag_vector_vector)) # new irrep!
```

```
matrix_vector_projected_matrix_vector
1.0
|C|   1,   6,   3,   8,   6
------------------------------
A1    1,   1,   1,   1,   1
T1    3,   1,  -1,   0,  -1
T2    3,  -1,  -1,   0,   1
```

In [19]:
```
# now look at diagonal part
matrix_diag_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                  [diagonal_projector])
test_representation(matrix_diag_vector_vector)
characters_diag_vector_vector = character_table_row(matrix_diag_vector_vector)
print(inner_product(characters_diag_vector_vector, characters_diag_vector_vector)) # reducible
```

```
matrix_vector_projected_matrix_vector
2.0
```

In [20]:
```
# only the traceless diagonal part
matrix_diag_traceless_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                  [diagonal_projector,traceless_projector])
test_representation(matrix_diag_traceless_vector_vector)
characters_diag_traceless_vector_vector = character_table_row(matrix_diag_traceless_vector_vector)
print(inner_product(characters_diag_traceless_vector_vector, characters_diag_traceless_vector_vector)) # irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
print("T2 ",fmt(characters_s_offdiag_vector_vector))
print("E  ",fmt(characters_diag_traceless_vector_vector)) # new irrep!
```

```
matrix_vector_projected_matrix_vector
1.0000000000000004
|C|   1,   6,   3,   8,   6
------------------------------
A1    1,   1,   1,   1,   1
T1    3,   1,  -1,   0,  -1
T2    3,  -1,  -1,   0,   1
E     2,   0,   2,  -1,   0
```

In [21]:
```
# what was the trace diagonal part?  sounds like trivial representaton
matrix_diag_trace_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                  [diagonal_projector,invert_projector(traceless_projector)])
test_representation(matrix_diag_trace_vector_vector)
characters_diag_trace_vector_vector = character_table_row(matrix_diag_trace_vector_vector)
print(inner_product(characters_diag_trace_vector_vector, characters_diag_trace_vector_vector)) # irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
print("T2 ",fmt(characters_s_offdiag_vector_vector))
print("E  ",fmt(characters_diag_traceless_vector_vector))
print("?? ",fmt(characters_diag_trace_vector_vector)) # A1 again
```

```
matrix_vector_projected_matrix_vector
1.0
|C|   1,   6,   3,   8,   6
------------------------------
A1    1,   1,   1,   1,   1
T1    3,   1,  -1,   0,  -1
T2    3,  -1,  -1,   0,   1
```

```
E      2,   0,   2,  -1,   0
??     1,   1,   1,   1,   1
```

In [29]:

```
# we still need one more, take direct product of two irreps and subtract all known irreps
# should not matter unless no new irrep is in product (T2 \otimes T2 does, e.g., not work)
def orthogonal_matrix(A, character_rows):
    P = A(ident) * 0.0
    for cr in character_rows:
        P += np.matrix(projection(A, cr))

    P = np.matrix(np.eye(len(P))) - P
    assert np.linalg.norm(P*P - P) < 1e-14

    def _mat(i):
        return P*A(i)*P
    _mat.__name__ = A.__name__ + "_projected"
    return _mat

matrix_last_irrep = orthogonal_matrix(
    #product_matrix(matrix_diag_traceless_vector_vector,matrix_diag_traceless_vector_vector),
    #product_matrix(matrix_s_offdiag_vector_vector, matrix_vector),
    product_matrix(matrix_vector,product_matrix(matrix_vector,matrix_vector)),
[
    characters_trivial,
    characters_vector,
    characters_s_offdiag_vector_vector,
    characters_diag_traceless_vector_vector
])

test_representation(matrix_last_irrep)
characters_last_irrep = character_table_row(matrix_last_irrep)
print(inner_product(characters_last_irrep, characters_last_irrep)) # irreducible!

print("|C|",fmt([len(c) for c in classes]))
print("-" * 30)
print("A1 ",fmt(characters_trivial))
print("T1 ",fmt(characters_vector))
print("T2 ",fmt(characters_s_offdiag_vector_vector))
print("E  ",fmt(characters_diag_traceless_vector_vector))
print("A2 ",fmt(characters_last_irrep)) # looks like an irrep!  we are done
```

```
matrix_vector_X_matrix_vector_X_matrix_vector_projected
1.0000000000000004
|C|    1,   6,   3,   8,   6
------------------------------
A1     1,   1,   1,   1,   1
T1     3,   1,  -1,   0,  -1
T2     3,  -1,  -1,   0,   1
E      2,   0,   2,  -1,   0
A2     1,  -1,   1,   1,  -1
```

Now we have the complete character table of O! This allows us to project the irreps out of any representation of G. This will be useful, when we construct operators to study the spectrum of H.

We have shown above, e.g., that $V_i = \bar\psi\gamma_i\psi$ transforms as a vector under O, i.e., they transform in the T1 representation.

We also have already seen in the last chapter that a bilinear V_i creates a meson-type state (made out of one quark and one anti-quark). If we now want to systematically study the spectrum of two suchs mesons, the above discussion would tell us that the operators

$$V_{ij}^{(2),\mathrm{T2}} = \frac{1}{2}(V_iV_j + V_jV_i)(1-\delta_{ij}) \tag{58}$$

transform in T2. Note that $(i,j) = (x,y), (x,z), (y,z)$ are all independent components of the three-dimensional T2 representation (check first column of the character table). Similarly we can create operators that transform in all other irreps and therefore study the complete spectrum of two-meson states.

Finally, we now include the reflections and complete this chapter by studying the full cubic group O_h.

In [30]:

```
elements = find_closure([
    lambda e: [interchange(b,0,1) for b in e],
    lambda e: [interchange(b,0,2) for b in e],
    lambda e: [interchange(b,1,2) for b in e],
    lambda e: [reflection(b,0) for b in e],
    lambda e: [reflection(b,1) for b in e],
    lambda e: [reflection(b,2) for b in e]
])

mt = multiplication_table(elements, basis)
ident = find_identity(elements, basis)
assert ident == 0 # for convenience always have first element be the identity
inv = inverse_list(mt, ident)

print(len(elements[0]))
classes = find_conjugation_classes(mt, inv, ident)
print(classes)
```

```
48
[{0}, {32, 1, 2, 4, 40, 19}, {33, 3, 35, 5, 39, 41, 21, 23}, {24, 12, 6}, {7, 8, 13, 16, 26, 28}, {9, 11, 45, 15, 47, 17, 27, 29}, {10, 43, 44, 46, 14, 25}, {18, 36, 30}, {34, 37, 38, 20, 22, 31}, {42}]
```

In [31]:

```
characters_trivial = character_table_row(matrix_trivial)
characters_determinant = character_table_row(matrix_determinant)
characters_vector = character_table_row(matrix_vector)

print(inner_product(characters_trivial, characters_trivial)) # == 1 -> irreducible
print(inner_product(characters_determinant, characters_determinant)) # == 1 -> irreducible
print(inner_product(characters_vector, characters_vector)) # == 1 -> irreducible
print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector)) # plus/minus for relative sign of first and last character of 1d
```

```
1.0
1.0
1.0
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
```

In [34]:

```
matrix_a_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                    [antisymmetric_projector])
test_representation(matrix_a_vector_vector)
characters_a_vector_vector = character_table_row(matrix_a_vector_vector)
print(inner_product(characters_a_vector_vector, characters_a_vector_vector)) # is irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector))
print("T1+",fmt(characters_a_vector_vector))
```

```
matrix_vector_projected_matrix_vector
1.0
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
T1+   3,  -1,   0,  -1,   1,   0,  -1,  -1,   1,   3
```

In [35]:

```
matrix_s_offdiag_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                    [symmetric_projector,invert_projector(diagonal_projector)])
test_representation(matrix_s_offdiag_vector_vector)
characters_s_offdiag_vector_vector = character_table_row(matrix_s_offdiag_vector_vector)
print(inner_product(characters_s_offdiag_vector_vector, characters_s_offdiag_vector_vector)) # is irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector))
print("T1+",fmt(characters_a_vector_vector))
print("T2+",fmt(characters_s_offdiag_vector_vector))
```

```
matrix_vector_projected_matrix_vector
1.0
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
T1+   3,  -1,   0,  -1,   1,   0,  -1,  -1,   1,   3
T2+   3,   1,   0,  -1,  -1,   0,   1,  -1,  -1,   3
```

In [36]:

```
# can find a partner irrep by multiplying any irrep with a 1d irrep
matrix_det_s_offdiag_vector_vector = product_matrix(matrix_s_offdiag_vector_vector, matrix_determinant)
test_representation(matrix_det_s_offdiag_vector_vector)
characters_det_s_offdiag_vector_vector = character_table_row(matrix_det_s_offdiag_vector_vector)
print(inner_product(characters_det_s_offdiag_vector_vector, characters_det_s_offdiag_vector_vector)) # is irreducible

print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector))
print("T1+",fmt(characters_a_vector_vector))
print("T2+",fmt(characters_s_offdiag_vector_vector))
print("T2-",fmt(characters_det_s_offdiag_vector_vector))
```

```
matrix_vector_projected_matrix_vector_X_matrix_determinant
1.0
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
T1+   3,  -1,   0,  -1,   1,   0,  -1,  -1,   1,   3
T2+   3,   1,   0,  -1,  -1,   0,   1,  -1,  -1,   3
T2-   3,  -1,   0,   1,  -1,   0,   1,  -1,   1,  -3
```

In [37]:

```
matrix_diag_traceless_vector_vector = projected_product_matrix(matrix_vector, matrix_vector,
                                                    [traceless_projector,diagonal_projector])
test_representation(matrix_diag_traceless_vector_vector)
characters_diag_traceless_vector_vector = character_table_row(matrix_diag_traceless_vector_vector)
print(inner_product(characters_diag_traceless_vector_vector, characters_diag_traceless_vector_vector)) # is irreducible

matrix_det_diag_traceless_vector_vector = product_matrix(matrix_diag_traceless_vector_vector, matrix_determinant)
test_representation(matrix_det_diag_traceless_vector_vector)
characters_det_diag_traceless_vector_vector = character_table_row(matrix_det_diag_traceless_vector_vector)
print(inner_product(characters_det_diag_traceless_vector_vector, characters_det_diag_traceless_vector_vector)) # is irreducible
print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector))
print("T1+",fmt(characters_a_vector_vector))
print("T2+",fmt(characters_s_offdiag_vector_vector))
print("T2-",fmt(characters_det_s_offdiag_vector_vector))
print("E+ ",fmt(characters_diag_traceless_vector_vector))
print("E- ",fmt(characters_det_diag_traceless_vector_vector))
```

```
matrix_vector_projected_matrix_vector
1.0000000000000004
matrix_vector_projected_matrix_vector_X_matrix_determinant
1.0000000000000004
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
T1+   3,  -1,   0,  -1,   1,   0,  -1,  -1,   1,   3
T2+   3,   1,   0,  -1,  -1,   0,   1,  -1,  -1,   3
T2-   3,  -1,   0,   1,  -1,   0,   1,  -1,   1,  -3
E+    2,   0,  -1,   2,   0,  -1,   0,   2,   0,   2
E-    2,   0,  -1,  -2,   0,   1,   0,   2,   0,  -2
```

In [38]:

```
matrix_last_irrep_minus = orthogonal_matrix(
    product_matrix(product_matrix(matrix_vector, matrix_vector), matrix_vector),
[
    characters_trivial,
    characters_determinant,
    characters_vector,
    characters_a_vector_vector,
    characters_s_offdiag_vector_vector,
    characters_diag_traceless_vector_vector,
    characters_det_s_offdiag_vector_vector,
    characters_diag_traceless_vector_vector,
    characters_det_diag_traceless_vector_vector
])

test_representation(matrix_last_irrep_minus)
characters_last_irrep_minus = character_table_row(matrix_last_irrep_minus)
print(inner_product(characters_last_irrep_minus, characters_last_irrep_minus)) # irreducible!

matrix_last_irrep_plus = product_matrix(matrix_last_irrep_minus, matrix_determinant)
test_representation(matrix_last_irrep_plus)
characters_last_irrep_plus = character_table_row(matrix_last_irrep_plus)
print(inner_product(characters_last_irrep_plus, characters_last_irrep_plus)) # is irreducible


print("|C|",fmt([len(c) for c in classes]))
print("-" * 60)
print("A1+",fmt(characters_trivial))
print("A1-",fmt(characters_determinant))
print("T1-",fmt(characters_vector))
print("T1+",fmt(characters_a_vector_vector))
print("T2+",fmt(characters_s_offdiag_vector_vector))
print("T2-",fmt(characters_det_s_offdiag_vector_vector))
print("E+ ",fmt(characters_diag_traceless_vector_vector))
print("E- ",fmt(characters_det_diag_traceless_vector_vector))
print("A2-",fmt(characters_last_irrep_minus))
print("A2+",fmt(characters_last_irrep_plus))

# And we have the complete character table with all irreps of the full cubic group!
```

```
matrix_vector_X_matrix_vector_X_matrix_vector_projected
1.0000000000000007
matrix_vector_X_matrix_vector_X_matrix_vector_projected_X_matrix_determinant
1.0000000000000007
|C|   1,   6,   8,   3,   6,   8,   6,   3,   6,   1
------------------------------------------------------------
A1+   1,   1,   1,   1,   1,   1,   1,   1,   1,   1
A1-   1,  -1,   1,  -1,   1,  -1,   1,   1,  -1,  -1
T1-   3,   1,   0,   1,   1,   0,  -1,  -1,  -1,  -3
T1+   3,  -1,   0,  -1,   1,   0,  -1,  -1,   1,   3
T2+   3,   1,   0,  -1,  -1,   0,   1,  -1,  -1,   3
T2-   3,  -1,   0,   1,  -1,   0,   1,  -1,   1,  -3
E+    2,   0,  -1,   2,   0,  -1,   0,   2,   0,   2
E-    2,   0,  -1,  -2,   0,   1,   0,   2,   0,  -2
A2-   1,   1,   1,  -1,  -1,  -1,  -1,   1,   1,  -1
A2+   1,  -1,   1,   1,  -1,   1,  -1,   1,  -1,   1
```

In [39]:

```
# We end this chapter by demonstrating the orthonormality of the irreps
characters = [
    characters_trivial,
    characters_determinant,
    characters_vector,
    characters_a_vector_vector,
    characters_s_offdiag_vector_vector,
    characters_det_s_offdiag_vector_vector,
    characters_diag_traceless_vector_vector,
    characters_det_diag_traceless_vector_vector,
    characters_last_irrep_minus,
    characters_last_irrep_plus
]

for i in range(len(characters)):
    for j in range(len(characters)):
        eps = abs(inner_product(characters[i],characters[j]) - (1 if i == j else 0))
        assert eps < 1e-13

# We can now use this character table to project out any irrep of any given product representation!
```

In [ ]:

```

```