6.S966: Practice Exam 1, Spring 2024

These are practice problems for Exam 1. They will likely take you longer than the actual exam as some of the problems are explicitly more tedious or difficult than what you will see in an exam setting. The hope is that these problems give you extra practice for exam-like questions and also encourage you to dive deeper into certain material. If you have questions - come to office hours and / or post on Piazza!

- This is a closed book exam. One page (8 1/2 in. by 11 in) of notes, front and back, are permitted. Calculators are not permitted.
- The total exam time is 1 hours and 20 minutes.
- The problems are not necessarily in any order of difficulty.
- Record all your answers in the places provided. If you run out of room for an answer, continue on a blank page and mark it clearly.
- If a question seems vague or under-specified to you, make an assumption, write it down, and solve the problem given your assumption.
- If you absolutely *have* to ask a question, come to the front.
- **Write your name on every piece of paper.**

Name: ______________________ MIT Email: ______________________

Name: ______________________________

Question	Points	Score
1	20	
2	20	
3	20	
4	20	
5	20	
Total:	100	

Name: ____________________

Sudoku for Group Theorists

1. (20 points) Complete the following multiplication and character tables below using the Rearrangement Theorem and the First and Second Wonderful Orthogonal Theorems for Character.

   (a) Complete the following multiplication table.

	0	1	2	3	4	5	6	7
0	____	4	____	6	2	3	0	5
1	4	2	5	0	7	____	1	3
2	7	5	6	4	3	1	____	0
3	6	____	4	5	1	7	3	2
4	2	7	3	____	5	0	4	6
5	3	6	1	7	0	2	5	4
6	0	1	2	3	____	5	6	____
7	5	3	0	2	6	4	7	1

   (b) Cyclic groups are only irreducible over complex numbers (rather than real numbers), but have all 1D irreps. Complete the table below for the cyclic group C_3 where A_1 is the trivial irrep and A_2 and A_3 are (complex) 1D representations. $\omega = e^{i2\pi/3}$ and $\omega^2 = e^{i4\pi/3} = e^{-i2\pi/3}$. Hint: $1 + \omega + \omega^2 = 0$.

	E	C_3	$(C_3)^2$
A_1	____	____	____
A_2	____	ω	ω^2
A_3	____	ω^2	____

Name: ______________________________

Parsing Proofs

2. (20 points) In this problem, we will present a single step of some of the proofs shown in class, exercises, or notes and ask what properties of matrices or groups or lemmas or theorem, allows us to take this step.

   (a) Part 1 of Schur's lemma states that if a square matrix M commutes with all the elements of an irreducible representation, then it must be of the form λI for some λ.

   The proof follows the following steps:

   1. Every representation is similar to one with only hermetian matrices
   2. if M computes with a set Hermetian matrices, so do $M1 = M + M^*$ and $M2 = i(M - M^*)$
   3. Each of $M1$ and $M2$ is hermitian. So they must be diagnolizable
   4. If a Matrix Mi commutes with a diagonal matrix D. Then either Mi has block diagonal form, or D is constant
   5. Every matrix in an irrep must be full rank.

   i. Let f be any class function, and D be an irreducible representation. $(f(g) = f(h^{-1}gh) \forall g, h \in \mathcal{G})$. Show that $X = \sum_{g \in \mathcal{G}} f(g)D(g)$ is a constant matrix.

   ii. Find the constant in the matrix above in terms of the character of the irreducible reprsentation and the function f

   iii. Argue if D is not reducible, then if we take a similarity transform giving it block diagonal form where each block is an irrep. Then each block of X would still be a constant matrix with the same constant you found above.

iv. Argue that the characters of the irreducible representations in the regular representation must span the space of all class functions.
Hint: consider the orthognal complement, and compute X in two ways.

(b) The Wonderful Orthogonal Theorem can be written in two ways.

$$\sum_R D^{(\Gamma_j)}_{\mu\nu}(R) D^{(\Gamma_{j'})}_{\nu'\mu'}(R^{-1}) = \frac{h}{l_j}\delta_{\Gamma_j\Gamma_{j'}}\delta_{\mu\mu'}\delta_{\nu\nu'} \tag{1}$$

$$\sum_R D^{(\Gamma_j)}_{\mu\nu}(R) [D^{(\Gamma_{j'})}_{\mu'\nu'}(R)]^* = \frac{h}{l_j}\delta_{\Gamma_j\Gamma_{j'}}\delta_{\mu\mu'}\delta_{\nu\nu'} \text{ if the representations are unitary.} \tag{2}$$

i. Use properties of unitary matrices to derive line (2) from line (1).

ii. We then proceeded to equate

$$\sum_R \sum_\mu D^{(\Gamma_j)}_{\mu\nu}(R) D^{(\Gamma_{j'})}_{\nu'\mu}(R^{-1}) = \sum_R \sum_\mu D^{(\Gamma_j)}_{\nu'\mu}(R^{-1}) D^{(\Gamma_{j'})}_{\mu\nu}(R) \tag{3}$$

Why were we able to reorder the representations in line (3)?

(c) In our proof of Case 2 of the Wonderful Orthogonality Theorem ($l_j = l_{j'}$ and $\Gamma_j = \Gamma_{j'}$) we arrived at the following equation:

$$c''_{\nu\nu'}\delta_{\mu\mu'} = \sum_R D^{(\Gamma_{j'})}_{\mu\nu}(R)D^{(\Gamma_{j'})}_{\nu'\mu'}(R^{-1}), \text{ where } c''_{\nu\nu'} = \frac{c}{c'_{\nu\nu'}} \tag{4}$$

We then chose $\mu = \mu'$ and summed over μ to start solving for $c''_{\nu\nu'}$

$$c''_{\nu\nu'}\sum_\mu \delta_{\mu\mu} = c''_{\nu\nu'}l_{j'} = \sum_R \sum_\mu D^{(\Gamma_{j'})}_{\mu\nu}(R)D^{(\Gamma_{j'})}_{\nu'\mu}(R^{-1}) \tag{5}$$

where $l_{j'}$ is the dimension of the $\Gamma_{(j')}$ representation.

i. Why were we allowed to make this choice ($\mu = \mu'$) and perform this sum over μ?

ii. We then proceeded to equate

$$\sum_R \sum_\mu D^{(\Gamma_{j'})}_{\mu\nu}(R)D^{(\Gamma_{j'})}_{\nu'\mu}(R^{-1}) = \sum_R \sum_\mu D^{(\Gamma_{j'})}_{\nu'\mu}(R^{-1})D^{(\Gamma_{j'})}_{\mu\nu}(R) \quad = \sum_R D^{(\Gamma_{j'})}_{\nu'\nu}(R^{-1}R) \tag{6}$$

Why were we able to reorder the representations in this expression?

(d) In class, we proved that group convolution with respect to group $\mathcal{G}$ is equivalent under it's action. In particular:

$$L_g(f) \star \psi = L_g(f \star \psi) \tag{7}$$

Where $L_g f(x, y) = (g^{-1} \circ f)(x, y) = f(g^{-1}x, g^{-1}y)$.

i. If the group $\mathcal{H} \subset \mathcal{G}$ is the group of symmetries of the image function $f(L_h(f) = f, \forall h \in \mathcal{H})$. Argue that the correlation (with respect to group $\mathcal{G}$) is invariant under H.

ii. In general, What subset of $\mathcal{G}$ is group convolution (with respect to $\mathcal{G}$) invariant under when the input image is symmetric under group $\mathcal{H}$? briefly explain your reasoning.

Name: ______________________________

Isomorphisms of Multiplication Tables of Order 8

3. (20 points) Below are five group multiplication tables of groups of order 8. Here, we are using various symbols instead of numbers, so it is clear in your answers which table you are giving answers to.

	a	b	c	d	f	g	h	j
a	c	f	j	h	d	a	b	g
b	f	c	d	g	j	b	a	h
c	j	d	g	b	h	c	f	a
d	h	g	b	c	a	d	j	f
f	d	j	h	a	g	f	c	b
g	a	b	c	d	f	g	h	j
h	b	a	f	j	c	h	g	d
j	g	h	a	f	b	j	d	c

Table 1

	α	β	γ	δ	ϵ	ζ	η	θ
α	θ	η	ζ	ϵ	δ	γ	β	α
β	η	θ	ϵ	ζ	γ	δ	α	β
γ	ζ	δ	θ	β	η	α	ϵ	γ
δ	ϵ	γ	η	α	θ	β	ζ	δ
ϵ	δ	ζ	β	θ	α	η	γ	ϵ
ζ	γ	ϵ	α	η	β	θ	δ	ζ
η	β	α	δ	γ	ζ	ϵ	θ	η
θ	α	β	γ	δ	ϵ	ζ	η	θ

Table 2

	①	②	③	④	⑤	⑥	⑦	⑧
①	①	②	③	④	⑤	⑥	⑦	⑧
②	②	①	④	③	⑥	⑤	⑧	⑦
③	③	④	①	②	⑦	⑧	⑤	⑥
④	④	③	②	①	⑧	⑦	⑥	⑤
⑤	⑤	⑥	⑦	⑧	①	②	③	④
⑥	⑥	⑤	⑧	⑦	②	①	④	③
⑦	⑦	⑧	⑤	⑥	③	④	①	②
⑧	⑧	⑦	⑥	⑤	④	③	②	①

Table 3

	○	□	△	◇	⬠	⬡	⯃	★
○	□	⬠	★	⯃	△	◇	○	⬡
□	⬠	△	⬡	○	★	⯃	□	◇
△	★	⬡	⯃	⬠	◇	□	△	○
◇	⯃	○	⬠	⬡	□	★	◇	△
⬠	△	★	◇	□	⬡	○	⬠	⯃
⬡	◇	⯃	□	★	○	△	⬡	⬠
⯃	○	□	△	◇	⬠	⬡	⯃	★
★	⬡	◇	○	△	⯃	⬠	★	□

Table 3

	♈	♉	♊	♋	♌	♍	♎	♏
♈	♌	♈	♍	♉	♋	♎	♏	♊
♉	♈	♉	♊	♋	♌	♍	♎	♏
♊	♏	♊	♌	♍	♎	♈	♉	♋
♋	♉	♋	♏	♌	♈	♊	♍	♎
♌	♋	♌	♎	♈	♉	♏	♊	♍
♍	♊	♍	♋	♎	♏	♌	♈	♉
♎	♍	♎	♉	♏	♊	♋	♌	♈
♏	♎	♏	♈	♊	♍	♉	♋	♌

Table 5

Name: ____________________

(a) Give the symbol that corresponds to the identity element for each table

(b) For each symbol, give it's inverse.

$a^{-1} \rightarrow$	$\alpha^{-1} \rightarrow$
$b^{-1} \rightarrow$	$\beta^{-1} \rightarrow$
$c^{-1} \rightarrow$	$\gamma^{-1} \rightarrow$
$d^{-1} \rightarrow$	$\delta^{-1} \rightarrow$
$f^{-1} \rightarrow$	$\epsilon^{-1} \rightarrow$
$g^{-1} \rightarrow$	$\zeta^{-1} \rightarrow$
$h^{-1} \rightarrow$	$\eta^{-1} \rightarrow$
$j^{-1} \rightarrow$	$\theta^{-1} \rightarrow$
①$^{-1} \rightarrow$	$\bigcirc^{-1} \rightarrow$
②$^{-1} \rightarrow$	$\square^{-1} \rightarrow$
③$^{-1} \rightarrow$	$\triangle^{-1} \rightarrow$
④$^{-1} \rightarrow$	$\diamond^{-1} \rightarrow$
⑤$^{-1} \rightarrow$	⬠$^{-1} \rightarrow$
⑥$^{-1} \rightarrow$	⬡$^{-1} \rightarrow$
⑦$^{-1} \rightarrow$	⯃$^{-1} \rightarrow$
⑧$^{-1} \rightarrow$	$\star^{-1} \rightarrow$
♈$^{-1} \rightarrow$	
♉$^{-1} \rightarrow$	
♊$^{-1} \rightarrow$	
♋$^{-1} \rightarrow$	
♌$^{-1} \rightarrow$	
♍$^{-1} \rightarrow$	
♎$^{-1} \rightarrow$	
♏$^{-1} \rightarrow$	

(c) What is the order of each element of a group? Reminder, the order of an element is how many times you must multiply the element by itself to return to the identity.

$a \to$	$\alpha \to$
$b \to$	$\beta \to$
$c \to$	$\gamma \to$
$d \to$	$\delta \to$
$f \to$	$\epsilon \to$
$g \to$	$\zeta \to$
$h \to$	$\eta \to$
$j \to$	$\theta \to$
①$\to$	◯ $\to$
②$\to$	□ $\to$
③$\to$	△ $\to$
④$\to$	◇ $\to$
⑤$\to$	⬠ $\to$
⑥$\to$	⬡ $\to$
⑦$\to$	⯃ $\to$
⑧$\to$	$\star \to$
♈$\to$	
♉$\to$	
♊$\to$	
♋$\to$	
♌$\to$	
♍$\to$	
♎$\to$	
♏$\to$	

(d) Are any of these groups isomorphic to one another? Explain your reasoning.

Name: ______________________________

A Group and its Characters

4. (20 points) In this problem you will generate a group and compute its character table.

   (a) Generate a group using the following 2D mirror and rotation.

$$\begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \text{Mirror across } y = \sigma_y$$

$$\begin{pmatrix} \cos\left(\frac{\pi}{2}\right) & -\sin\left(\frac{\pi}{2}\right) \\ \sin\left(\frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2}\right) \end{pmatrix} = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad \text{Counterclockwise rotation by } \frac{\pi}{2} = C_4^{(1)}$$

   Express all elements as 2x2 matrices and label each operations using the following notation e for the identity, C_2 for in-plane two-fold rotations, $C_4^{(j)}$ for in-plane counterclockwise four-fold rotations where j is 1 or 3 (Note, $C_2 = C_4^{(2)}$, i for 2D inversion, $\sigma_{x=y}$ for a mirror across the line $x = y$, $\sigma_{x=-y}$ for a mirror across the line $x = -y$, σ_x for a mirror across the line x, and σ_y for a mirror across the line y.

   (b) This group has 5 conjugacy classes. Give the sets of elements in each conjugacy class. For example if both $C_4^{(1)}$ and $C_4^{(3)}$ form a conjugacy class, then the conjugacy class is the set $\{C_4^{(1)}, C_4^{(3)}\}$.

(c) Compute the trace of the 2D rotation and mirror representations you generated in Part (a) for each conjugacy class. This representation is irreducible. Label which trace belongs to which class.

(d) Using the constraint $\sum_j l_j^2 = h$ where l_j is the dimension of the j^{th} irrep and h is the order (size) of the group, determine the dimensions of the irreps of this group.

(e) Complete the character table below. Γ_1 is the trivial representation and Γ_2 is the irrep that transforms as your 2D representation of 2D mirrors and rotations from part (a).

- Label the conjugacy classes as the sets of elements in the conjugacy class. For example, label the conjugacy class of $C_4^{(1)}$ and $C_4^{(3)}$ as $\{C_4^{(1)}, C_4^{(3)}\}$. The ordering of the classes does not matter, but using the same order as the instructions for part (a).
- Label the missing irreps as Γ_1', Γ_1'', ... for other 1D irreps and Γ_2', Γ_2'' ... for 2D irreps, as needed. You do not need to worry about absolute ordering, only that the irreps are labeled by the correct dimension.
- Give the characters for the missing irreps using the Wonderful Orthogonality Theorem for Character and what you know about the characters for Γ_1 and Γ_2.

	e	______	______	______	______
Γ_1	1	1	1	1	1
Γ__	______	______	______	______	______
Γ__	______	______	______	______	______
Γ__	______	______	______	______	______
Γ_2	2	______	______	______	______

Name: ______________________________

Interpreting Outputs

5. (20 points) In the following questions, we will present you code snippets using the functions you have coded in the exercises and ask you to interpret what the outputs means. You may assume the following has been imported.

```
import numpy as np
from symm4ml import groups, group_conv, linalg, rep, vis
import torch
```

(a) Try running the following, but it's taking a long time to evaluate.

```
groups.generate_group(
    np.array([
        [ 0.99994517, -0.01047178 ],
        [ 0.01047178,  0.99994517 ]
    ]).reshape(1, 2, 2)
)
```

Why is this code taking long to evaluate? Explain your reasoning.

(b) You run the following code snippet and it returns the following output.

```
p3_matrices = groups.permutation_matrices(3)
linalg.infer_change_of_basis(p3_matrices, p3_matrices)
>> array([[[ 5.77350269e-01,  0.00000000e+00,  0.00000000e+00],
        [ 0.00000000e+00,  5.77350269e-01,  0.00000000e+00],
        [ 0.00000000e+00,  0.00000000e+00,  5.77350269e-01]],

       [[-3.70074342e-17,  4.08248290e-01,  4.08248290e-01],
        [ 4.08248290e-01,  7.40148683e-17,  4.08248290e-01],
        [ 4.08248290e-01,  4.08248290e-01, -3.70074342e-17]]])
```

Name: ______________________________

i. Explain what is happening in line (2).

ii. What does the output produced by line (2) tell us about the representation of the 3D permutation matrices? Explain your reasoning.

(c) Suppose we have the following multiplication tables saved into `table_dresselhaus` and `table_perm`, respectively.

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	4	5	2	3
2	2	5	0	4	3	1
3	3	4	5	0	1	2
4	4	3	1	2	5	0
5	5	2	3	1	0	4

D_3 Mult. Table

	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	0	3	2	5	4
2	2	4	0	5	1	3
3	3	5	1	4	0	2
4	4	2	5	0	3	1
5	5	3	4	1	2	0

$P(3)$ Mult. Table

What does the following output tell us about these two multiplication tables?

```
groups.isomorphisms(table_dresselhaus, table_perm)
>> {(0, 1, 2, 5, 3, 4),
 (0, 1, 5, 2, 4, 3),
 (0, 2, 1, 5, 4, 3),
 (0, 2, 5, 1, 3, 4),
 (0, 5, 1, 2, 3, 4),
 (0, 5, 2, 1, 4, 3)}
```

Explain what the output means in the context of the rows and columns of the multiplication tables above.

(d) In the following snippet, we perform the eigenvalue decomposition of M.

```
M = np.array([[[1, 0, 0], [0, 2, 0], [0, 0, 2]]])
np.linalg.eigh(M)
>> EighResult(
    eigenvalues=array([[1., 2., 2.]]),
    eigenvectors=array([[[1., 0., 0.],
        [0., 1., 0.],
        [0., 0., 1.]]]))
```

Where the eigenvalues are `eigenvalues` and the eigenvectors are given as the columns of `eigenvectors`. Use the re-express M as a sum of projector matrices created using the eigenvectors multiplied by the appropriate eigenvalue. Simplify your expression so that you only have one projector per unique eigenvalue.

Name: ______________________________

symm4ml Docstring listing

Modules listed in order: `groups`, `group_conv`, `linalg`, `rep`

```
groups.conjugacy_classes:
Returns the conjugacy classes of the group.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        Set of conjugacy classes. Each conjugacy class is a set of integers.

groups.factor_group:
Returns the factor group of the group.
    Input:
        table: np.array of shape [n, n] where entries correspond to indices of
            group elements.
        selfconj_sub: set of indices for self-conjugate subgroup.
    Output:
        Multiplication table of factor group of order n2 as sets of elements of
            the group
        np.array sets of ints of shape [n2, n2]
        Multiplication table of factor group in terms of indices of right cosests
        np.array of shape [n2, n2] where entries correspond to indices of first
            dim of matrices.

groups.factors:
Returns the set of factors of n.
    Input:
        n: int
    Output:
        Set of integers that divide n.
    Example:
        factors(12) = {1, 2, 3, 4, 6, 12}

groups.generate_group:
Generate new group elements from matrices (group representations)
    Input:
        matrices: np.array of shape [n, d, d] of known elements
        decimals: int number of decimals to round to when comparing matrices
    Output:
        group: np.array of shape [m, d, d], where m is the size of the resultant
            group

groups.identity:
Returns the index of the identity element.
    Input:
```

```
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        Index of identity element.
    Raises:
        ValueError("No or multiple identities") if there is no or multiple
            identities.

groups.inverse_permutation:
Inverts a permutation.
    Input:
        p: np.array of shape [n], a permutation of the integers 0, ..., n-1
    Output:
        np.array of shape [n], the inverse permutation of p

groups.inverses:
Returns the indices of the inverses of each element.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        np.array of shape [n] where the ith entry is the index of the inverse of
            the ith element.
    Raises:
        ValueError("Every element does not have one inverse") if there is no or
            multiple inverses.

groups.is_associative:
Tests whether the multiplication table is associative.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        True if the table represents an associative binary operation, False
            otherwise.

groups.is_closed:
Tests whether the multiplication table is closed.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        True if the table represents a closed binary operation, False otherwise.

groups.isomorphisms:
Finds all isomorphisms between two multiplication tables of same order.
```

```
    Input:
        table_src: np.array of shape [n, n] where the entry at [i, j] is the index
             of the product of the ith and jth elements in the source group.
        table_dst: np.array of shape [n, n] where the entry at [i, j] is the index
             of the product of the ith and jth elements in the destination group.
    Output:
        A set of isomorphisms encoded as tuples ``h`` of length ``n``.
        Each element ``h[i]`` is the index of the image of the ith element in the
            source group.

groups.left_coset:
Returns the left coset of the ith element.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
        subgroup_indices: Indices of elements in the subgroup.
    Output:
        Set of left cosets for each element in the group. Each coset is
            represented as a set of indices.

groups.make_multiplication_table:
Makes multiplication table for group.
    Input:
        matrices: np.array of shape [n, d, d], n matrices of dimension d that form
             a group under matrix multiplication.
        tol: float numberical tolerance
    Output:
        Group multiplication table.
        np.array of shape [n, n] where entries correspond to indices of first dim
            of matrices.

groups.permutation_matrices:
Generates all permutation matrices of n elements
    Input:
        n: int
    Output:
        matrices: np.array of shape [n!, n, n]

groups.permute_mul_table:
Multiplication table of the same group with a different ordering.
    Tip: If your solution does not work, try with the inverse permutation.
    Input:
        table: np.array of shape [n, n]
        perm: np.array of shape [n]
    Output:
        permuted multiplication table.
```

```
        np.array of shape [n, n]

groups.right_coset:
Returns the right coset of the ith element.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
        subgroup_indices: Indices of elements in the subgroup.
    Output:
        Set of right cosets for each element in the group. Each coset is
            represented as a frozenset of indices.
    Example:
        right_coset(np.array([[0, 1], [1, 0]]), {0}) == {frozenset({1}), frozenset
            ({0})}

groups.subgroups:
Find all subgroups of group.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Output:
        Yields tuples of elements that form subgroup.

groups.surjective_homomorphisms:
Finds all surjective homomorphisms from one group to another.
    Input:
        table_src: np.array of shape [n_src, n_src] where the entry at [i, j] is
            the index of the product of the ith and jth elements in the source
            group.
        table_dst: np.array of shape [n_dst, n_dst] where the entry at [i, j] is
            the index of the product of the ith and jth elements in the destination
             group.
    Output:
        A set of surjective homomorphisms encoded as tuples ``h`` of length ``
            n_src``.
        Each element ``h[i]`` is the index of the image of the ith element in the
            source group.

groups.test_group:
Tests whether the multiplication table is valid.
    Input:
        table: np.array of shape [n, n] where the entry at [i, j] is the index of
            the product of the ith and jth elements in the group.
    Raises:
        ValueError("Invalid indices") if the table contains invalid indices (is
            not closed).
```

```
        ValueError("No or multiple identities") if the table does not contain
            exactly one identity.
        ValueError("Every element does not have one inverse") if not every element
             has an inverse.
        ValueError("Not associative") if the table is not associative.

group_conv.image2D_group_convolution:
Performs group convolution of inputs and filters over the regular representation
    Input:
        rep_2D: torch.Tensor of shape [|G|, 2, 2] of the group representation as 2
            D rotations and mirrors
        rep_reg: torch.Tensor of shape [|G|, |G|, |G|] of the left regular
            representation
        inverse: torch.LongTensor of shape [|G|] with the indices of the inverses
            of the group elements
        input: torch.Tensor of shape [batch, channel_in, rep_reg_in, height, width
            ]
        filter: torch.Tensor of shape [channel_out, channel_in, rep_reg_filter,
            kernel_height, kernel_width]
    Output:
        output: torch.Tensor of shape [batch, channel_out, rep_reg_out]

group_conv.image2D_group_convolution_filter_bank:
Creates rotated filter bank for 2D image convolution
    Input:
        rep_2D: torch.Tensor of shape [|G|, 2, 2] of the group representation as 2
            D rotations and mirrors
        filter: torch.Tensor of shape [channel_out, channel_in, rep_reg_filter]

group_conv.rep_reg_group_convolution:
Performs group convolution of inputs and filters over the regular representation
    Input:
        rep_reg: torch.Tensor of shape [|G|, |G|, |G|] of the left regular
            representation
        input: torch.Tensor of shape [batch, channel_in, rep_reg_in]
        filter: torch.Tensor of shape [channel_out, channel_in, rep_reg_filter]
    Output:
        output: torch.Tensor of shape [batch, channel_out, rep_reg_out]

linalg.gram_schmidt:
Return the Gram-Schmidt orthonormalization of the vectors.
    Input:
        vectors: an (n1, d) matrix of n1 complex vectors of dimension d
        tol: a tolerance for the zero vector
    Output:
        Q: an (n2, d) matrix of n2 orthonormal vectors, with n2 <= n1
```

```
        P: a (d, d) projector onto the span of the orthonormal vectors in Q

linalg.infer_change_of_basis:
Compute the change of basis matrix from X1 to X2.
    tip: Use the function nullspace
    Input:
        X1: an (n, d1, d1) array of n (d1, d1) matrices
        X2: an (n, d2, d2) array of n (d2, d2) matrices
    Output:
        Sols: An (m, d1, d2) array of m solutions.
        Each solution is a (d1, d2) matrix that satisfies X1 @ S = S @ X2.

linalg.nullspace:
Return the nullspace of the matrix.
    Input:
        matrix: an (n, d) matrix of n complex vectors of dimension d
        tol: a tolerance for the zero eigenvalue
    Output:
        Q: an (m, d) matrix containing orthogonal vectors spanning the nullspace (
            obtained by Gram-Schmidt)
        P: a (d, d) projector onto the span of the nullspace

linalg.orthogonal_complement:
Return orthogonal vectors spanning the orthogonal complement of the span of the
    input vectors.
    Input:
        vectors: an (n1, d) matrix of n1 complex vectors of dimension d
        tol: a tolerance for the zero vector
    Output:
        Q: an (n2, d) matrix of n2 orthonormal vectors spanning the orthoganl
            complement, with d - n1 <= n2 <= d
        P: a (d, d) projector onto the orthogonal complement of the input vectors

linalg.projector:
Return the projector onto the vector v.
    Input:
        v: a d dimensional complex vector
    Output:
        P: a rank 1 matrix such that P @ v = v

rep.are_isomorphic:
Checks if representations are isomorphic.
    Input:
        rep1: np.array [n, d, d] representation of group. rep1[i] is a matrix that
            represents i-th element of group.
        rep2: np.array [n, d, d] representation of group. rep2[i] is a matrix that
```

```
        represents i-th element of group.
    You can assume that rep1 and rep2 are valid group representations.
  Output:
    True if representations are isomorphic.

rep.check_orthogonality_theorem:
Checks orthogonality theorem for a set of input representations.
  Input:
    irreps: List of representations, np.arrays of shape [n, d, d], where n is
        the order of group and d is the dimension of the representation. Not
        necessarily irreducible!
  Output:
    True if the theorem holds (i.e. the representations in the list are
        irreducible, unitary and pairwise orthogonal and have the appropriate
        self-inner product), False otherwise.

rep.direct_sum:
Computes direct sum of two representations.
  Input:
    rep1: np.array [n, d1, d1] representation of group. rep[i] is a matrix
        that
        represents i-th element of group.
    rep2: np.array [n, d2, d2] representation of group. rep[i] is a matrix
        that
        represents i-th element of group.
    You can assume that rep1 and rep2 are valid group representations.
  Output:
    Direct sum of representations. np.array [n, d1 + d2, d1 + d2].

rep.is_a_representation:
Checks if rep is a representation of the group represented by a given
  multiplication table.
  Input:
    table: np.array [n, n] where table[i, j] = k means i * j = k.
    rep: np.array [n, d, d] describing a possible representation of the group.
         rep[i] is a matrix corresponding to the action of the i-th element of
        the group.
  Output:
    True if rep is a representation.

rep.is_an_irrep:
Checks if rep is an irreducible representation of group represented by
  multiplication table.
  Input:
    table: np.array [n, n] where table[i, j] = k means i * j = k.
    rep: np.array [n, d, d] representation of group. rep[i] is matrix that
```

```
        represents i-th element of group.
    Output:
        True if rep is an irreducible representation.

rep.regular_representation:
Returns regular representation for group represented by a multiplication table.
    Input:
        table: np.array [n, n] where table[i, j] = k means i * j = k.
    Output:
        Regular representation. array [n, n, n] where reg_rep[i, :, :] = D(i) and
            D(i)e_j = e_{ij}.
                            Equivalently, D(g) |h> = |gh>
```

Name: ____________________

Work space

Name: ______________________________

Work space