Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF MATHEMATICS

MSCI RESEARCH PROJECT

Equivariant Machine Learning

Author:
Viba Courtney-Battista
01847210

Supervisors:
Kevin Webster, Jeroen Lamb,
Victoria Klein

Submitted in partial fulfillment of the requirements for the MSci in Mathematics at Imperial College London

June 10, 2024

Abstract

This thesis provides a generalised group theoretic approach to designing the framework for symmetry-aware models which respect equivariance and invariance to the underlying symmetry, or equivalently models that give predictable outcomes when the data is acted on by a predetermined symmetry. This is made possible in two key ways, first providing a method to develop the building blocks of an invariant model intuitively and with mathematical rigour. Second is to demonstrate the construction of a synthetic spherical dataset for models with Icosahedral symmetry.

Acknowledgments

I would like to extend my gratitude to my supervisors, Kevin Webster, Jeroen Lamb, and Victoria Klein for engaging me in such a captivating area of mathematics, and to Thibault Bertrand for his help and advice over the course of the project. I am grateful for my peers, Michael Pristin, Niamh Arthur, Joshua Rose, Sebastian Dixon and Arnav Singh for keeping me from mathematics induced insanity, even Lebesgue could not measure my gratitude. I would also like to thank Kartik Iyer and Kshitiz Acharya for studying group representation alongside me for this thesis. Finally, I would like to thank Bhavana Daryanani for teaching me how to use a thesaurus, without you this thesis would have been ~~very bad~~ really bad.

Plagiarism statement

The work contained in this thesis is my own work unless otherwise stated.

Signature: Viba Courtney-Battista
Date: June 10, 2024

Contents

Chapter 1

Introduction

“The chief forms of beauty are order and symmetry and definiteness, which the mathematical sciences demonstrate in a special degree..."

— Aristotle, Toward an Aesthetics of Blindness: An Interdisciplinary Response to Synge, Yeats, and Friel p.213

In the context of an increasingly data driven world, taking advantage of symmetries inherent in datasets is crucial for enhancing machine learning frameworks to improve generalisation and performance. The challenge of creating equivariant models is significant, and naturally involves assigning symmetries to groups from which we can begin to construct the architecture. This paper will restrict the discussion to finite groups.

1.1 Symmetries in nature - why we care

Symmetry is abundant in nature, being especially prevalent in the realms of mathematics and statistics, physics, and chemistry. Many examples include molecular structure, physical dynamical systems in nature such as weather, mechanics and population dynamics, and modelling man made scenarios and games.

When considering the problem of modelling these systems, the goal is to understand the situation in an efficient and correct manner. Hence, it becomes crucial to consider these symmetries in a meaningful way, ensuring the model accurately reflects our prior knowledge of the nature of the problem.

1.2 Machine learning with symmetry

This paper aims to provide the theory, framework and a rigorous example of how to tackle machine learning with symmetry. The study of group theory beautifully captures real world symmetries and dynamics, which is the backbone of invariant theory.

This theory is fundamental for current problems in computer vision, designing drugs, classification of images on nontrivial shapes, such as the sphere, which is relevant in the context of self driving cars or drones that take spherical input data.

A brief description of what it means for a model to be invariant, which will be expanded on in more depth in the mathematical background section is as follows:

Invariance: a model is invariant if its output is equivalent with respect to symmetry in the

data. Applying a symmetry can be thought of by "acting" on the input with an element in the corresponding symmetry group. For example if we are modelling an even function over the real numbers, we should square the input before training, this way the model cannot distinguish between x and -x, and thus gives the same output for both, f(x) = f(-x). In this example the symmetry group is a single reflection and is described by $\mathbb{Z}_2$.

Equivariance: a model is equivariant if a symmetry-transformed input provides a corresponding symmetry-transformed output. The corresponding example is as above but when modelling odd functions, where f(x) = -f(-x).

In the context of machine learning, python was used to generate code. This github page github.com/vibabattista/Equivariant-Machine_Learning details how to design symmetrical datasets, and construct the building blocks necessary for creating invariant or equivariant models, this is especially helpful for those who have not covered the mathematical theory, or those who need a dataset that abides by certain symmetries, as these datasets are a rarity. A framework for applying the theory to computer algebra packages is appropriately explained in "Computer Algebra Methods for Equivariant Dynamical Systems"[1] which forms a sturdy basis for the methods used, namely the use of invariant and equivariant polynomials at the core of a learnable model.

1.2.1 Existing approaches: "Geometric deep learning"

Exploring existing implementations of equivariant and invariant machine learning models reveals the evolution of different approaches and techniques. Initially the naive approach was prevalent, where models were trained on en extended dataset consisting of the original data and all symmetry-transformations of the data. This was to ensure models can recognise patterns irrespective of the known symmetry, a portrayal of this is is image classification where a cat is a cat whether it is in the centre of the image, the edge, or upside down. However this method is clear to be inefficient as it handles unnecessarily large datasets proportional to the number of symmetries leading to significant resource consumption. This sparked motivation for convolutional neural networks which offered significant improvements to translation invariant models. These developments in invariant machine learning encouraged more advanced concepts such as group convolutional neural networks, researched by Cohen and Welling in "Group Equivariant Convolutional Networks".[2], and equivariant multi layer perceptrons, researched by Marc Finzi "A Practical Method for Constructing Equivariant Multilayer Perceptrons for Arbitrary Matrix Groups"[3], in which he combined graph neural networks and group CNNs, have a more refined ability to capture a broader range of symmetries efficiently, aligning with the philosophy of the deeper methods discussed in this paper.

These are all very different approaches to the same problem. The theory in this paper is most similar to that of GCNNs as the underlying philosophy is to build a general method for constructing invariant models for a general compact group.

1.3 Invariant theory

Invariant theory is the bridge between symmetries in the real world to group theory. It helps find meaningful ways to express invariant and equivariant polynomials with respect to the structure of a group.

Within the study of groups and rings is Hilbert's Finiteness theorem, from which the study of fundamental invariants and equivariant polynomials flourished. These polynomials are immediately applicable to infusing mathematical models with invariance: in the context of machine learning, an invariant/equivariant model can be achieved through learning a function of the

fundamnetal invariants and equivariants. We can immediately find direct applications to modelling real world systems so long as these polynomials can be feasibly obtained. This gives a clear path to follow and allows us to begin implementing them.

So far a brief introduction to the concepts and intuition behind the main goal of building equivariant models for general symmetries has been provided, later these ideas will be expanded on, defined rigorously and combined to achieve a solid framework for building a model.

To enhance my understanding of group representation theory, which was unfamiliar to me prior to this thesis, I engaged with a variety of resources. The foundational knowledge was critical for understanding invariant theory and developing the contents of this thesis. I found that familiarising myself with simple analogies, such as the even and odd functions, enabled a smooth transition into invariant theory, and I suggest the reader to also consider these examples when engaging with the less accessible aspects of group representation theory.

If we carefully follow the steps introduced in this chapter to a group, such as the icosahedral group, the goal of classification of images on a sphere invariant to icosahedral rotation will no longer seem so far fetched.

1.3.1 Machine learning

An easily understandable stepping stone into fully invariant models is the concept of convolution, to transform the image into a more "feature-rich" space, by mapping small sections of the original image into this new space. This can be thought of as sweeping a transformation matrix across the image to transform the entire image, which allows the model to "forget" position and subsequently become entirely translation invariant.

We are trying to achieve invariance through a more generalised method, however it is instrumental to be aware of methods such as convolution when planning new invariant architectures. An invariant or equivariant model can be adapted for many uses, be more or less complex than the method provided here, or even be combined with other popular architectures that may make use of invariant layers.

The model architecture formulated here is the simplest form for an invariant or equivariant function, for this reason it should be considered as an alternative to a standard neural network with no such invariance, and should be viewed as a basis for more complex models that incorporate invariance in their structure.

This introduction has outlined the foundational concepts and motivations behind the study of equivariant machine learning through applications of group theory. We will continue to explore the mathematical underpinnings and the reasoning behind the benefits of invariant theory. The subsequent chapters will contain a detailed discussion bridging the theory to the applications of invariant modelling, including a conveniently designed synthetic dataset designed for a model with the rotational invariance of an icosahedron that can be readily applied.

Chapter 2

Mathematical Background

Understanding invariant and equivariant models begins with the task of rephrasing invariant functions as a functions of pre-calculated invariant polynomials. The invariant polynomials inherently exhibit invariance, allowing for a template for an invariant function. We later show that any function that takes the invariant polynomials as inputs is invariant by construction.

A logical question to ask is how do we find these polynomials? We delve deeper into group representation theory to answer this. Representation theory, in essence, involves representing each group element as a matrix. Given the complexity and size of many groups, manual calculation of the polynomials and certain representations is impractical, hence we will use established computational algorithms for computing the invariant polynomials provided by Karin Gatermann in "Gröbner bases, invariant theory and equivariant dynamics"[1], and utilise the SymPy package in python to develop an algorithmic method for obtaining the representations.

This chapter aims to rigorously define the concepts introduced alongside examples crafted to be intuitive to those without prior knowledge on the subject. The subsequent chapter will focus on applying the theory to finding a representation for the icosahedral group hands-on.

2.1 Group Theory

Definition 1. *A **Group** is a non empty set, G, equipped with a binary relation,$\cdot$, such that $(G, \cdot)$ satisfies the following group axioms:*

1. *Group Closure:* $\forall a, b \in G,\ a \cdot b \in G$
2. *Existence of identity element, e, such that* $\forall a \in G, e \cdot a = a \cdot e = a$
3. *Existence of inverses:* $\forall a \in G, \exists b \in G$ *such that* $a \cdot b = b \cdot a = e$
4. *Associativity:* $\forall a, b, c \in G, a \cdot (b \cdot c) = (a \cdot b) \cdot c$

Example A_5 is the group we work with throughout this paper. A_5 is the group of even permutations on a finite set, and is synonymous with the rotational symmetries on an icosahedron. As we can define A_5 via rotational symmetry, showing it is a group becomes trivial.

To be able to work with such a group it makes sense to ask how it can be represented in a form we are familiar with. Group representation theory studies the possible embeddings into the general linear group, for example we can describe the non-trivial action of $\mathbb{Z}_2$ in $\mathbb{R}$ by the matrix -1, or even in $\mathbb{R}^n$ by $-\mathbb{I}_n$, and the identity element to be $\mathbb{I}_n$. Checking we have a group with binary operation matrix multiplication is immediate.

Definition 2. *A **Representation** of a group G on a vector space V is a group homomorphism from G to the general linear group over V.*
Equivalently, it is a map ρ such that

$$\rho : G \to GL(V)$$

and

$$\rho(a \cdot b) = \rho(a)\rho(b), \forall a, b \in G$$

Example An easy example of a representation of $\mathbb{Z}_2$ is -1 and 1 in $GL(\mathbb{R})$, where 1 is identity and -1 is order two, and its own inverse, hence clearly forms a group. Furthermore to demonstrate the homomorphic property we can see that $\rho(a)\rho(b) = \rho(ab), \forall a, b \in \mathbb{Z}_2$.

Definition 3. ***Invariance** and **equivariance**: Consider a function $f : V \to W$, and a representation of a group G $\rho : G \to GL(V)$ We say f is ρ-**invariant** if*

$$f(\rho(g)x) = f(x), \forall g \in G, \forall x \in V$$

*Now consider a second representation $\rho_2 : G \to GL(W)$. f is ρ_1-ρ_2-**equivariant** if*

$$f(\rho_1(g)x) = \rho_2(g)f(x), \forall g \in G, \forall x \in V$$

Example $\mathbb{Z}_2$, represented by {1,-1} in $GL(\mathbb{R})$ has invariant polynomial $f = x^2$. Here, $f : \mathbb{R} \to \mathbb{R}$, so $\rho_1 = \rho_2 = \rho$ can act on the range and the domain. The case for $\rho(g) = 1$ is trivial, for $\rho(g) = -1$ we have $(-x)^2 \equiv x^2$ and we are done. An equivariant function for this representation is x^3, again the 1 case is trivial, and for -1 we have $(-x)^3 \equiv -(x^3)$ and we are done.

Furthermore, every even function is a function of x^2, $f_{even} = f(x^2)$ and similarly for every odd function: $f_{odd} = xf(x^2)$

The invariant polynomials form a subring of the polynomial ring on V. Later we will see that this subring is finitely generated.

Definition 4. ***Subrepresentation**: Let V, W be vector spaces and $W \subset V$. For a representation (V,ρ), a **subrepresentation** is any representation $(W, \rho|_W)$.*

Example Consider $\mathbb{R} \subset GL(\mathbb{R}^2)$. Given a representation

$$\rho : \mathbb{Z}_2 \to GL(\mathbb{R}^2)$$

sending e to the identity and the second element to the negative identity, there exists a restriction of ρ onto $\mathbb{R}$ such that

$$\rho|_{\mathbb{R}} : \mathbb{Z}_2 \to GL(\mathbb{R})$$

where rho sends the elemets to 1 and -1 as above. Here, $(\mathbb{R}, \rho|_{\mathbb{R}})$ is the subrepresentation of ρ.

A representation can generally be decomposed into a direct sum of subrepresentations. Understanding subrepresentations allows us to simplify and study groups by decomposing them into more manageable representations, a necessary step for dealing with complex symmetries computationally.

Definition 5. ***Irreducible representation.*** *a Representation that has no subrepresentations except {0} and itself.*

We will refer to irreducible representations as irreps.

Definition 6. ***Maschke's Theorem.*** *For a representation ρ of a finite group G over a field with characteristic not dividing $|G|$, if ρ has a subrepresentation, ρ_1, then it has another subrepresentation ρ_2 such that $\rho = \rho_1 \oplus \rho_2$ [4]*

The implications of Maschke's theorem guarantee a decomposition into irreps for finite groups over $\mathbb{C}$, or more conveniently for Gatermann's algorithms, $\mathbb{R}$. This is due to $\mathbb{C}$ and $\mathbb{R}$ having characteristic zero. The implementations in this paper need not consider other fields, hence the simplified Maschke's theorem is sufficient: finite groups admit a decomposition into a direct sum of real irreps.

A cornerstone in invariant theory is Hilbert's finiteness theorem, which states that the ring of invariant polynomials is finitely generated.

Theorem 1. ***Hilbert's Finiteness Theorem.*** *Let G be a compact group acting on a finite dimensional vector space V. The ring of invariant polynomials on V, denoted $V[x]_\rho \subset V[x]$, is finitely generated.*

Example Any invariant polynomial $f : V \to \mathbb{R}$ can be parameterised as:

$$f(x) = p(\pi_1(x), ...\pi_{N_{inv}})$$

for some set of ρ-invariant polynomials $\{\pi_1, ...\pi_{N_{inv}}\}$ on V, and polynomial $p : \mathbb{R}^{N_{inv}} \to \mathbb{R}$.

In machine learning, Hilbert's finiteness supports the concept of learning an invariant function by learning a general function of the polynomial invariants. Extending Hilbert's finiteness from polynomials to general continuous functions is made clear in the following section.

Definition 7. *The **character** of a representation ρ is defined as $\chi : G \to \mathbb{C}$ such that*

$\chi_\rho(g) = tr(\rho(g))$ and

1. $\chi_{\rho_1 \oplus \rho_2} = \chi_{\rho_1} + \chi_{\rho_2}$
2. $\chi_{\rho_1 \otimes \rho_2} = \chi_{\rho_1}\chi_{\rho_2}$
3. $\chi_\rho(1) = n$ *if ρ has degree n*
4. $\chi(g^{-1}) = \chi(g)$ *for all $g \in G$*
5. $\chi(h^{-1}gh) = \chi(g)$ *for all $g, h \in G$*

Characters provide a simple way to deal with representations without directly dealing with matrices. They become vital for constructing the irreps of a group by verifying trace of the matrix.

Example The character table for A_5:

A useful property of the character is that representations that share their character are equivalent. Each irrep corresponds to a conjugacy class, of which A_5 has 5.

Class	1	(12)(34)	(123)	(12345)	(13524)
Size	1	15	20	12	12
χ_1	1	1	1	1	1
χ_2	3	-1	0	$\frac{1+\sqrt{5}}{2}$	$\frac{1-\sqrt{5}}{2}$
χ_3	3	-1	0	$\frac{1-\sqrt{5}}{2}$	$\frac{1+\sqrt{5}}{2}$
χ_4	4	0	1	-1	-1
χ_5	5	1	-1	0	0

Table 2.1 Irreducible character table

2.1.1 Invariant theory

Theorem 2. ***Weierstrass approximation theorem****: Let f be a continuous valued function on $\mathbb{R}$, then f can be uniformly approximated by polynomials. [5]*

When paired with the Weierstrass approximation theorem, Hilbert's finiteness theorem now extends to continuous functions, a key result for invariant machine learning. The means for approximating any continuous invariant function is now accessible, achieved by learning a function of the invariant polynomials.

It can futher be shown that a similar anzats for equivariant polynomials is as follows:

$$f(x) = \sum_{i=0}^{N_{eq}} F_i(x) p_i(\pi_1, ...\pi_{Ninv})$$

for a generating set of equivariant polynomials denoted F_i, invariant polynomials π_i, and polynomials p_i, which is due to the fact if f is equivariant and π is invariant then $f(x)\pi(x)$ is equivariant also.[6]

Theorem 3. ***Weyl****. Let $V = \bigoplus_\alpha V_\alpha^{m_\alpha}$ and $V' = \bigoplus_\alpha V_\alpha^{dimV_\alpha}$ be two vector spaces acted on by G such that $m_\alpha \geq dim(V_\alpha)$. Then the space of polynomial invariants on V is linearly spanned by $\{f \circ \mathbf{A}_{f \in F}\}$, where $\mathbf{A} : V \to V'$, $\mathbf{A} = \bigoplus_\alpha \mathbb{I}_{V_\alpha} \bigotimes A_\alpha$ and $A_\alpha : V_\alpha^{m_\alpha} \to V_\alpha^{dim(V_\alpha)}$[7].*

Weyl's theorem provides the means to parameterise invariant functions from arbitrary vector spaces using the same set of invariant polynomials. This is beneficial for practical use as it enables modelling with invariance to some group without the hassle of calculating invariants. Instead of computing new invariants, the trick is to find the transformations A_t and compose with the known invariants.

2.1.2 Application to machine learning

Machine learning revolves around the concept that any continuous function can be modelled using a standard universal architecture, a neural network.

Definition 8. *A **Neural Network** is a function $f : \mathbb{R}^n \to \mathbb{R}^m$ defined by a composition of affine transformations alternating with a non-linear activation function, ReLU$(x) = max\{x, 0\}$.*

Definition 9. ***Universal Approximation Theorem of neural networks****. For any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$ there exists a sequence of neural networks $\hat{f}_n : \mathbb{R}^n \to \mathbb{R}^m$ such that $\hat{f}_n$ converges uniformly to f as $n \to \infty$.[8]*

Hilbert's finiteness theorem and **Weierstrass' approximation theorem** provide a form for a G-invariant continuous function on the vector space V, as a sequence of polynomials. The claim that we can learn this function with standard neural networks is concluded by direct application of the **Universal Approximation Theorem** to a function of the invariant polynomials of G on V. Intuitively, for a general invariant function f this can be seen as:

1. Approximate f to an arbitrary precision by a polynomial, p. (Weierstrass)
2. Parameterise p in terms of invariant polynomials, $p = \hat{p}(\pi_1, ...\pi_n)$ (Hilbert)
3. Learn $\hat{p}$ to an arbitrary precision with a neural network, $\mathcal{N}$. (Universal Approximation)
4. Now $\mathcal{N}(\pi_1, ...\pi_n)$ is invariant and approximates f to an arbitrary precision, concluding the claim.

Whilst this is a remarkable advancement, it relies on calculating the polynomials for a fixed vector space V. Fortunately, Weyl's theorem can aid in generalising the method. We have already seen that we can recycle the known invariants for use on new vector spaces by transforming the inputs by A_t, now we can consider introducing the A_t's as learnable parameters, drastically reducing computational complexity and allowing the model to focus on the most valuable polynomials that best capture the invariance of the system. Now it is more practical to calculate the polynomials once for a simplistic space and generalise them to new spaces. Hence, by Weyl's theorem, can write any function invariant on a space $\hat{V}$ in terms of invariant polynomials on V through the transformation $\pi \circ A_t$ as:

$$f(x) = \sum_{t=1}^{T} \pi_t(A_t x) \qquad (\star)$$

As a general outline to constructing an invariant model we can do the following:

- For a group representing a specific symmetry, find the real irreps of the group.
- Apply Gatermann's algorithm to the group elements under each of the irreps to find the invariant polynomials. For some spaces this can be very difficult or time consuming, in these cases we obtain the polynomials in |G| variables and apply Weyl's theorem.
- Denote the polynomials with π as above. Set the A_t's in the form above as learnable linear functions.
- Use the form for f in $(\star)$. Recall learning f is equivalent to learning the f_t's, which need not be invariant, with standard neural networks.
- Train the model. This is inherently invariant by construction.

Chapter 3

The Icosahedral group

The following chapter addresses the preparation of the icosahedral group for the application Gatermann's algorithm, a fundamental step in obtaining the invariant polynomials. To do this we need to find all irreps of the icosahedral group and write every element in this irrep. This task is non-trivial and should be done by hand, the computational aspect can assist with matrix multiplication at the end.

I will begin by introducing the group of icosahedral symmetries, A_5. I have chosen to work with A_5 as it is the largest finite subgroup of the group of symmetries on a sphere, SO(3). It can act as an approximation of SO(3), therefore acting a solution to more practical problems, such as invariant classification of images on a sphere.

To see how the theory can be applicable to real high definition images we consider Weyl's theorem. Through the application of Weyl's theorem we are able build towards an arbitrarily fine discretisation of a sphere by building the polynomials on a permutation representation in 60 variables.

Definition 10. *The Icosahedral group, A_5, is the rotation group of an icosahedral, which is one of five platonic solids characterised by 20 triangular faces 12 vertices and 30 edges. An element of A_5 can be viewed as a map from the icosahedron to itself. It can also be defined through its presentation: $A_5 : \langle g, h | g^2, h^5, (gh)^3 \rangle$, in other words it consists of 2, 3 and 5-cycles and is a simple subgroup of S_5, the full permuatation group on 60 points. It is useful to know that $|A_5| = 5!/2 = 60$.*

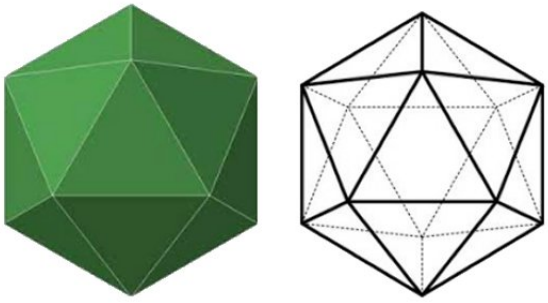

Figure 3.1 The Icosahedron. The elements of A_5 corresponds to every rotation of this shape.

A_5 is a finite group, hence there is a well known algorithm for computing fundamental invariants and equivariants as detailed in 'Gröbner bases, invariants theory and equivariant dynamics' by Karin Gatermann.[1] This algorithm requires the elements of A_5 to be written in its irreducible representations, hence the goal is now to obtain the irreps of A_5.

3.0.1 The regular representation of A_5

I suggest following the provided code for this section here, as a demonstration of following the method in SymPy.

The regular representation is a reducible representation that contains each of the irreps at least once in a direct sum, the multiplicity of each irrep is equal to its dimension. Hence to obtain the regular representation of A_5 we must first find every irreducible representation of A_5.

The non-computational methods for finding the irreps of A_5 are non-trivial, and detailed in the appendix. The computational methods are valid, but inefficient in time complexity. This subsection will outline the initial method attempted to obtain the irreps.

The dimensionalities of the irreps are related to $|A_5|$ as follows: $\sum_i d_i^2 = |A_5|$ [7]

It can be readily checked that the only feasible solution to this is 1, 3, 3, 4 and 5. Since these numbers correspond to the dimension of each irrep and the dimension of the regular representation is 60 we have a form for our regular representation.

3.0.2 Calculating the irreps

We have now calculated the form of the regular representation:

$$1 \oplus M^3_{3\times3} \oplus \hat{M}^3_{3\times3} \oplus M^4_{4\times4} \oplus M^5_{5\times5}$$

where $M_{n\times n}$ corresponds to an irrep, note the third irrep is distinct from the second.

The permutation representation, 60×60 permutation matrices, is isomorphic to this as it contains the entire behaviour of the group, and it can therefore be attained through a linear transformation. In order to achieve this we must first find the permutation representation by observing the relationships between the group elements.

This is easily done in two steps, the first is to arbitrarily order the elements, then compose each to each other element of A_5, resulting in a 60×60 matrix of indices from 1 to 60 (Figure 3.2), corresponding to the chosen ordering. Now the $(i, j)^{th}$ element of this matrix corresponds to index for the element $g_i \cdot g_j \in A_5$. The second step is to extract the permutation representation from this.

The permutation representation is exactly as it sounds, for an element g its representation is intuitively described by how it acts with all other elements. For constructing the matrix for the g_i^{th} element we compose g_i with g_j yielding one unique element, $g_k \in A_5$, we then put a one at the k^{th} index of the j^{th} row and zero for the remainer of the row.

We now have the problem well phrased and the matrices ready to be solved. Let P_i denote the permutation representation for element i and M denote the regular representation of block diagonal irreps (a matrix of variables to be found). Since we know there exists a unique representation in this form we have that the equation $Q^{-1}PQ = M$ is well phrased for some change of basis matrix Q.

When calculating the irreps in python we can define M as a matrix of SymPy variables and then solve the system of equations for the variables in M. Unfortunately our matrices are very large, and current methods to solve matrices with unknown variables increases factorially with the number of unknown variables in the determinant. It is infeasible to expect these equations to be solved in a timely manner, even for one element let alone two generators of A_5, so instead we will turn our attention to new more mathematically technical methods to find the irreps. The reader is encouraged to build upon this computational method by reading

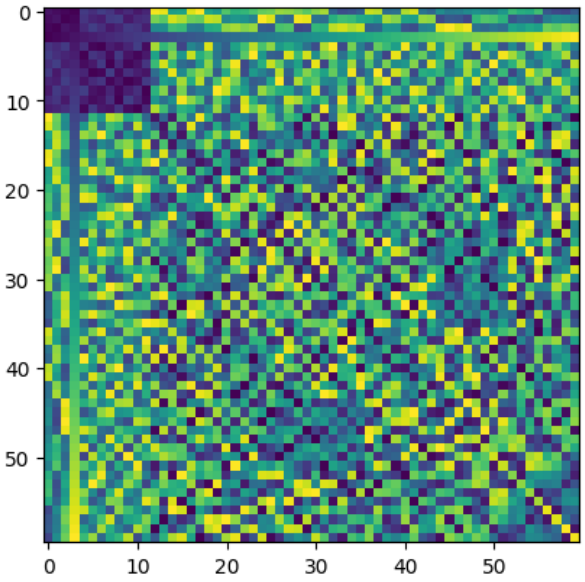

Figure 3.2 The composition matrix, each element is an index from 1 to 60. Note it is not a symmetric matrix despite the nice symmetric nature of the conjugacy classes. The identity element can be immediately found to be at index 3 as this row is increasingly linearly from 1 to 60.

the SymPy documentation[1] to improve the algorithm, as finding irreps for arbitrary groups is non-trivial without using this method.

[1] https://docs.sympy.org/latest/guides/solving/index.html

Creating invariant and equivariant models for general finite groups depends heavily on a generalised computational method to obtaining the irreps of a group (and of course the fundamental invariants, which we have algorithms for).

Turning our attention back to representation theory, we can attempt to find each irrep by hand, which is done in a variety of ways.

3.1 The irreps of A_5

So far I have provided a method of obtaining the irreps using SymPy. However this is far too computationally expensive to be useful, so here I will detail multiple mathematical methods instead.

This section inherently contains advanced character theory and group representation theory. My formal studies did not include group representation, motivating an intuitive and instructional guide that requires no background knowledge. For the more technical group representation techniques, sources have been included for further reading. If instead the reader is only interested in computational applications and using the irreps this can also be read as a guide to constructing them. It is recommended to follow the python notebook provided here[2]. This code can be edited to tailor for different groups, different representations or run as it is set up to generate the real irreps for the elements of A_5.

Recall the irreducible character table introduced in 2.2, each character corresponds to an irrep. For each irrep, the group can be divided into its conjugacy classes. This is because each element in a conjucay class under the irreducible representation corresponding to χ_i will be a matrix with a fixed trace.

This is the key takeaway from the character table, keep in mind that every element of the group belongs to a conjugacy class, and each representation will have the same trace, given by the corresponding entry in the irreducible character table.

Example Consider the rotation matrix on $\mathbb{R}^3$ corresponding to π rotation about the x-axis:

$$R_x(\pi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

We see the trace of this matrix is -1, and it its square is the identity. Hence this corresponds with an element of the conjugacy class of (12)(34) (even 2-cycles) and could be the second or third irrep as its character is its trace.

This is just one element under the second irrep, calculating 60 elements for 5 irreps would be time consuming. Instead recall a presentation of A_5 is

$$\langle g, h | g^2, h^5, (gh)^3 \rangle$$

Using this we only need to find the irreps of a 2-cycle, g, and a 5-cycle, h, such that $(gh)^3 = e$. It is readily checked that g = (12)(34) and h = (12345) yields $(gh)^3 = e$ so I will use these to calculate the irreps. Finally, every irrep can be found constructed in the github code shared in the appendix and footnote.

[2] https://github.com/vibabattista/Equivariant-Machine_Learning/blob/master/irrep_generation.ipynb

The next section is split by character, or equivalently I will calculate the irreps in order of size.

3.1.1 χ_1

This character is the trivial character, meaning the trace of our one dimensional matrices must be identically 1 for all elements. Hence we have (1) for each element in A_5.

3.1.2 χ_2

Here both characters correspond with three by three matrix representations, for the 2-cycle of interest we see from the character table that the character is constant and -1 across both. For the 5-cycle of interest we see it is ϕ and $-\phi^{-1}$ where $\phi = \frac{1+\sqrt{5}}{2}$.

There is no generalised method to calculate an irrep from an irreducible character, so it suffices to find the 3×3 representations by hand, then check they correspond to the irreducible character. Fortunately we are already familiar with how the group acts in three dimensions so we can find the irreps geometrically.

Recall the definition of A_5, the group of icosahedron rotations. The most concise way to define a rotation in $\mathbb{R}^3$ is to define an axis of rotation, and an angle of rotation about that axis. Geometrically we can see the following:

- Any 5-cycle corresponds to a rotation with axis through opposing vertices, by an angle of $\frac{2\pi}{5}$. These axes are relatively nice to work with as the x, y and z coordinates of the unit axis are the permutations of $(\pm 1, \pm\phi, 0)$.
- Any 3-cycle corresponds to a rotation with axis through opposing faces, by an angle of $\frac{2\pi}{5}$.
- Any 2-cycle corresponds to a rotation with axis through opposing edges, by an angle of π.

χ_2 is one of the more tedious characters to produce irreps for as it requires a labelling of the icosahedron into its five conjugacy classes, considering we are searching for two distinct generators it is vital the irreps in each dimension correspond to that exact element, despite the similarities with elements in its conjugacy class. Even though all elements in a conjugacy class have the same order, and the irreps are independent, the regular representation which is the block diagonal matrix of irreps will then not necessarily match with an element in A_5.
Specifically, even though the irreps are of the correct order, the regular representation constructed will fail to satisfy $(ab)^3$.

Bearing this in mind, it may be more convenient to obtain an "easy" to find irrep and then construct the irrep for (12)(34) through matrix multiplication.

(12345)

To start, $(12345) \in A_5$ is order five and corresponds to rotation about $(1, \phi, 0)$ with angle $\frac{2\pi}{5}$. Letting x, y and z be the unit axis vector and θ be the angle we can find the irrep using the axis/angle matrix below.

$$\begin{bmatrix} \cos(\theta) + x^2(1-\cos(\theta)) & xy(1-\cos(\theta)) - z\sin(\theta) & y\sin(\theta) + xz(1-\cos(\theta)) \\ z\sin(\theta) + xy(1-\cos(\theta)) & \cos(\theta) + y^2(1-\cos(\theta)) & yz(1-\cos(\theta)) - x\sin(\theta) \\ xz(1-\cos(\theta)) - y\sin(\theta) & x\sin(\theta) + yz(1-\cos(\theta)) & \cos(\theta) + z^2(1-\cos(\theta)) \end{bmatrix} \tag{3.1}$$

Hence the first irrep in $GL(\mathbb{R}^3)$ is:

$$\rho_2((12345)) = \begin{bmatrix} \frac{1}{2} & -\frac{1}{4}+\frac{\sqrt{5}}{4} & \frac{1}{4}+\frac{\sqrt{5}}{4} \\ -\frac{1}{4}+\frac{\sqrt{5}}{4} & \frac{1}{4}+\frac{\sqrt{5}}{4} & -\frac{1}{2} \\ -\frac{\sqrt{5}}{4}-\frac{1}{4} & \frac{1}{2} & -\frac{1}{4}+\frac{\sqrt{5}}{4} \end{bmatrix} \tag{3.2}$$

With trace ϕ as required from the irreducible character table.

(12)(34)

We need to be careful to ensure that the 2-cycle we represent in this section is exactly (12)(34). When we defined the rotation for (12345) we assigned 1,2,3,4 and 5 to specific points on the icosahedron as in figure 3.3.

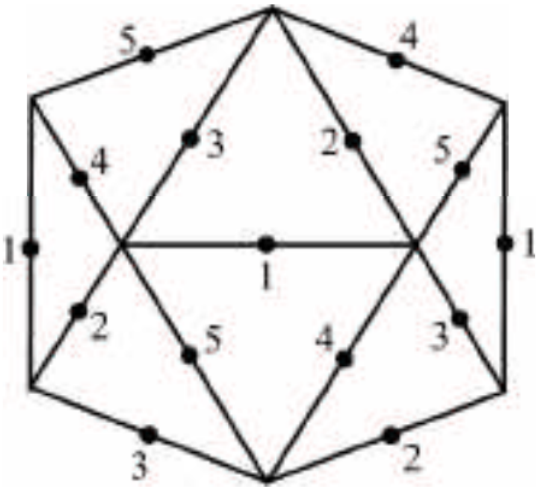

Figure 3.3 Arbitrary numbering of conjugacy classes on unit icosahedron. x-axis horizontal, z-axis vertical, y-axis perpendicular.[9]

To ensure we obtain the correct element, note that $(12)(34) = g^4(23)(45)g$, where g = (12345), a known element. Thankfully, (23)(45) corresponds to rotation by π about the y axis, hence

$$\rho_2((23)(45)) = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.3}$$

Last, we compute $\rho_2((12)(34)) = \rho_2((12345))^4\rho_2((23)(45))\rho_2((12345))$ to obtain:

$$\rho_2((12)(34)) = \begin{bmatrix} -\frac{\sqrt{5}}{4}-\frac{1}{4} & \frac{1}{2} & \frac{1}{4}-\frac{\sqrt{5}}{4} \\ \frac{1}{2} & -\frac{1}{4}+\frac{\sqrt{5}}{4} & -\frac{\sqrt{5}}{4}-\frac{1}{4} \\ \frac{1}{4}-\frac{\sqrt{5}}{4} & -\frac{\sqrt{5}}{4}-\frac{1}{4} & -\frac{1}{2} \end{bmatrix} \tag{3.4}$$

3.1.3 χ_3

This character also corresponds to a representation in $GL(\mathbb{R}^3)$. Referring back to the character table we see for (12)(34) the character is -1 for both χ_2 and χ_3, whereas for (12345) the character for χ_2 corresponds with the character χ_3 for the element (13524). An intuitive guess would be the representation is constant across the characters for (12)(34) and the third irrep for (12345) is equal to the second irrep for (13524). This intuition is backed up by the fact an automorphism composed with a representation is a representation [9]. Define an automorphism of A_5 to be conjugation by (12), it can be checked that this sends (12345) to (13524) and fixes (12)(34). Now we can state that $\rho_3((12)(34)) = \rho_2((12)(34))$.

To find the third irrep for (12345) we can find the second irrep for (13524) by a similar method as we did for (12345). The axis of rotation for this element is $(-1, \phi, 0)$ by an angle of $\frac{2\pi}{5}$, which yields:

$$\rho_3((12345)) = \rho_2((13524)) = \begin{bmatrix} -\frac{\sqrt{5}}{4} - \frac{1}{4} & \frac{1}{2} & \frac{1}{4} - \frac{\sqrt{5}}{4} \\ \frac{1}{2} & -\frac{1}{4} + \frac{\sqrt{5}}{4} & -\frac{\sqrt{5}}{4} - \frac{1}{4} \\ \frac{1}{4} - \frac{\sqrt{5}}{4} & -\frac{\sqrt{5}}{4} - \frac{1}{4} & -\frac{1}{2} \end{bmatrix} \tag{3.5}$$

3.1.4 χ_4

χ_4 corresponds to the 4-dimensional irreps. First note that there is a 5-dimensional representation corresponding to the permutation representation, as A_5 is the even permutations on 5 elements. Then it can be found that this representation is reducible, and by an application of Maschke's Theorem we can decompose into the trivial irrep and a 4×4 irrep. For finding the exact form for these matrices see "The Irreducible Representations of A5" by M. Wesslén[9].

$$\rho_4((12345)) = \begin{bmatrix} 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \tag{3.6}$$

$$\rho_4((12)(34)) = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

3.1.5 χ_5

As an initial approach to finding the fifth irrep of A_5 we can attempt to induce a representation of A_5 from a representation of a subgroup of A_5, namely, A_4. Again, the details for finding the exact matrix form using this method have been spared, as we will focus on an alternative method to obtain a real representation. The details can be found in the bibliography [9]. The irreps given in this paper are

$$\rho_5((12)(34)) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \omega^2 & 0 \\ 0 & 0 & \omega & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.8}$$

$$\rho_5((12345)) = \begin{bmatrix} 0 & 0 & 0 & 0 & \omega \\ 0 & 0 & \omega^2 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ \omega & 0 & 0 & 0 & 0 \\ 0 & \omega^2 & 0 & 0 & 0 \end{bmatrix} \tag{3.9}$$

Where ω is the third root of unity. This may be a valid irreducible representation however the existing algorithms for creating invariant polynomials rely on the representation mapping to $GL(\mathbb{R}^n)$ and we have complex valued matrices.

A solution worth pursuing is finding a change of basis that make both matrices real, which does not seem unrealistic when both matrices are constructed from 1s and roots of unity. For example $\rho_5((12)(34))$ yields real eigenvalues, so diagonalising provides a real matrix. $\rho_5((12345))$ on the other hand diagonalises as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{4} + \frac{\sqrt{5}}{4} - i\sqrt{\frac{\sqrt{5}}{8} + \frac{5}{8}} & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{4} + \frac{\sqrt{5}}{4} + i\sqrt{\frac{\sqrt{5}}{8} + \frac{5}{8}} & 0 & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{5}}{4} - \frac{1}{4} - i\sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}} & 0 \\ 0 & 0 & 0 & 0 & -\frac{\sqrt{5}}{4} - \frac{1}{4} + i\sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}} \end{bmatrix}$$

(3.10)

Which of course can be viewed as a block diagonal matrix, $1 \otimes Q \otimes P$ where Q and P are two dimensional. Since both Q and P are diagonal with a complex conjugate pairs as entries, there exists a transformation to make this matrix real:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & -\frac{1}{4} + \frac{\sqrt{5}}{4} & -\sqrt{\frac{\sqrt{5}}{8} + \frac{5}{8}} & 0 & 0 \\ 0 & \sqrt{\frac{\sqrt{5}}{8} + \frac{5}{8}} & -\frac{1}{4} + \frac{\sqrt{5}}{4}, & 0 & 0 \\ 0 & 0 & 0 & -\frac{\sqrt{5}}{4} - \frac{1}{4} & -\sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}} \\ 0 & 0 & 0 & \sqrt{\frac{5}{8} - \frac{\sqrt{5}}{8}} & -\frac{\sqrt{5}}{4} - \frac{1}{4} \end{bmatrix}$$

(3.11)

At this point we have a realisation for one of the generators, applying the same transformation to the other will likely yield another real representation, or at least give guidance to find the simultaneous realisation transformation for the generators. Calculating the inverse eigenvector matrix for this transformation in SymPy is extremely computationally expensive and it becomes infeasible to compute the transformation, so despite this matrix corresponding with past literature [10] I attempted a new method.

Since we know that the permutation representation is not irreducible we can turn to various literature on the topic to find a real representation. In "The Basis Functions and the Matrix Representations of the Single and Double Icosahedral Point Group" by Koun Shirai 1992[11] the irreps are derived in a basis such that the fifth irrep is always real. Unfortunately, as pointed

out by Lisa L. Everett and Alexander J. Stuart in 2009 in "Icosahedral (A5) Family Symmetry and the Golden Ratio Prediction for Solar Neutrino Mixing"[12], there are typos in the matrices provided. Fortunately, after applying a multitude of various corrections, both a 2-cycle and a 3-cycle were recovered from this paper. Provided below is the 2-cycle and 5-cycle under the 5×5 representation of S_5 respectively.

$$\mathrm{B} = \frac{1}{2}\begin{bmatrix} \frac{1-3\phi}{4} & -\frac{\phi^2}{2} & -\frac{1}{2\phi^2} & -\frac{\sqrt{5}}{2} & \frac{\sqrt{3}}{4\phi} \\ \frac{\phi^2}{2} & -1 & 1 & 0 & \frac{\sqrt{3}}{2\phi} \\ \frac{1}{2\phi^2} & 1 & 0 & -1 & \frac{\sqrt{3}\phi}{2} \\ -\frac{\sqrt{5}}{2} & 0 & 1 & 1 & \frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{4\phi} & -\frac{\sqrt{3}}{2\phi} & -\frac{\sqrt{3}\phi}{2} & \frac{\sqrt{3}}{2} & \frac{3\phi-1}{4} \end{bmatrix} \quad \mathrm{A} = \frac{1}{2}\begin{bmatrix} \frac{1-3\phi}{4} & \frac{\phi^2}{2} & -\frac{1}{2\phi^2} & \frac{\sqrt{5}}{2} & \frac{\sqrt{3}}{4\phi} \\ \frac{\phi^2}{2} & 1 & 1 & 0 & \frac{\sqrt{3}}{2\phi} \\ -\frac{1}{2\phi^2} & 1 & 0 & -1 & -\frac{\sqrt{3}\phi}{2} \\ \frac{\sqrt{5}}{2} & 0 & -1 & 1 & -\frac{\sqrt{3}}{2} \\ \frac{\sqrt{3}}{4\phi} & \frac{\sqrt{3}}{2\phi} & -\frac{\sqrt{3}\phi}{2} & -\frac{\sqrt{3}}{2} & \frac{3\phi-1}{4} \end{bmatrix}$$

We know these matrices correspond to 2 and 5-cycle elements. However these matrices are dangerous to work with as the 2-cycle may live in S_5 but not A_5, for example (12), which is an odd permutation. From here we can test in SymPy whether the 2-cycle is odd or even with this quick check: compute conjugation of powers of B (this will always result in a 2-cycle), if any of these new matrices of the form $A_n = B^{5-n}AB^n$ satisfy the relation $(A_nB)^3 = e$ then we know we have found a two cycle that generates A_5 and hence is even.

Following the code provided on github it is clear that the sign of A is odd and therefore not an element of A_5. To create an element in A_5 we can use a 3-cycle to construct an even two cycle. This is fairly effortless through trial and error as the chance of finding the correct order of odd 2-cycles, 3-cycles and the known 5-cycle is high. Through the use of SymPy the relation that yields an even 2-cycle is ACB^2 where c represents the 3-cycle, and can be found from Shirai 1992 [11]. The even 2-cycle matrix given after computing ACB^2 is

$$\begin{bmatrix} -\frac{1}{16}+\frac{3\sqrt{5}}{16} & \frac{3}{8}-\frac{\sqrt{5}}{8} & -\frac{\sqrt{5}}{4} & \frac{\sqrt{5}}{8}+\frac{3}{8} & \frac{\sqrt{3}}{16}+\frac{\sqrt{15}}{16} \\ \frac{3}{8}-\frac{\sqrt{5}}{8} & \frac{1}{2} & 0 & -\frac{1}{2} & \frac{\sqrt{3}}{8}+\frac{\sqrt{15}}{8} \\ -\frac{\sqrt{5}}{4} & 0 & \frac{1}{2} & \frac{1}{2} & \frac{\sqrt{3}}{4} \\ \frac{\sqrt{5}}{8}+\frac{3}{8} & -\frac{1}{2} & \frac{1}{2} & 0 & -\frac{\sqrt{3}}{8}+\frac{\sqrt{15}}{8} \\ \frac{\sqrt{3}}{16}+\frac{\sqrt{15}}{16} & \frac{\sqrt{3}}{8}+\frac{\sqrt{15}}{8} & \frac{\sqrt{3}}{4} & -\frac{\sqrt{3}}{8}+\frac{\sqrt{15}}{8} & \frac{1}{16}-\frac{3\sqrt{5}}{16} \end{bmatrix} \text{ with } \mathrm{C} = \frac{1}{2}\begin{bmatrix} -1 & 0 & 0 & 0 & -\sqrt{3} \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ \sqrt{3} & 0 & 0 & 0 & -1 \end{bmatrix}$$

Finally, we have an even 2-cycle and a 5-cycle to generate A_5! For completeness the generators are ACB^2 and B, with traces equal to their corresponding irreducible characters as required.

Note that (12)(34) is in the same conjugacy class as (23)(45), implying they share a character. This means the even 2-cycle may not coincide with (12)(34). This is not an issue as the elements can be matched after construction, however it is important to note that the irreps cannot be "stacked" into a regular representation when constructing the group.

3.1.6 Generating irreps for all elements

Gatermann's algorithm requires the irreps for each element, with five irreps per element and 60 elements that comes to 300 irreps to find! Thankfully, from the previous sections, we are set up to construct every element under every irrep from the irreps of the generators under the presentation for A_5.

Computationally, important points to consider include:

- Floating point errors. Constructing the entire group results in multiplying the error up to an upper bound of 60 times, so this method would require high depth floating point precision.
- Time complexity. Constructing the group through naively checking all combinations results in an extremely long and inefficient algorithm, instead it is helpful to use the known identities the generators satisfy.

The algorithm governing the make group function is outlined below. The underlying strategy is basic: post multiply each element in the queue by a and b, check if it is already in the group (by construction of the inputs every matrix created will be in the final group by closure of groups), if it is not in the group then add this element to the queue and repeat, else skip this element. The queue is initialised with the generators a and b, and the group is initialised with a, b, and e, the identity matrix.

Algorithm 1: Make Group from Generators Function

Input: Two generators, an initial queue with the generators, an initial group of the identity and the generators, all as matrices

Output: Whole matrix group generated by recursive multiplications

Initialize: group_label $\leftarrow [' e',' a',' b']$
Initialize: queue_label $\leftarrow [' a',' b']$
Function *make_group_from_generators*
  if *len(queue)* $= 0$ **then**
    return group
  end
  while *len(queue)* > 0 ***and*** *len(group)* < 62 **do**
    $a \leftarrow$ queue.pop(0);
    $a\_label \leftarrow$ queue_label.pop(0);
    for $i, g \in$ *enumerate(generators)* **do**
      if *(*$a\_label[-1] =' a'$ ***and*** $i = 0$*)* ***or*** *(*$a\_label[-4:] =' bbbb'$ ***and*** $i = 1$*)* ***or*** $a\_label[-4:] =' ababa'$ ***and*** $i = 1$ **then**
        continue to next iteration
      end
      $new \leftarrow$ sp.Matrix$(a \times g)$;
      if ***not***(*any*(*sp.Eq*$(new, item)$*for item in group*)) **then**
        group.append(new);
        group_label.append$(a_gen +$ group_label$[i+1])$;
        queue_label.append$(a_gen +$ group_label$[i+1])$;
        queue.append(new);
        print(len(group));
        print(group_gen[-1]);
      end
    end
  end
  return group
end

Being cautious with SymPy is key, we should keep all matrices exact to ensure the error is zero, however this makes checking matrix equality slow and dubious. Keep in mind that "==" in SymPy returns True for exact equality of form, for example $\sqrt{2}\sqrt{3} == \sqrt{6}$ returns False. Instead what should be used is "Eq" which returns True if the numerical value, or symbol form is equivalent. If this becomes too slow, one alternative is to compute the new element in SymPy,

then convert to NumPy and check against a pre-calculated NumPy version of the group which is updated alongside the exact group. This enables us to check to an arbitrary precision without the time penalty of using Eq, however, depending on the size of the group, converting every element to NumPy floating point numbers may be even more time consuming.

To make it more efficient and much clearer to see what is going on, the first generator in the input should be the 2-cycle, the second the 5-cycle, as this is the order used to create a pseudo group containing strings exactly tracking the order in which the generators are applied for any element, decomposing it into a sequence of the two generators. This allows the enforcing of the identities whilst avoiding computing SymPy variable matrix multiplication, which becomes exponentially slower. As an example, midway through the group generation the next element to be checked is a matrix, this matrix will correspond to a string 'abbbabb', which represents the element in terms of the generators. This is helpful because now we can simply check the string for 'aa's, 'bbbbb's and 'ababab's which we know are identity, without computing any matrix multiplication.

By applying this algorithm to the irreps for the generators found in the previous section we can obtain the irreps for every element in A_5.

The appendix details alternative methods to obtain the irreps.

Chapter 4

Implementation

This chapter is motivated by the challenge of training and testing models with icosahedral invariance, due to the lack of simple datasets with specific symmetries. As discussed prior, the icosahedral group will serve as an approximation of SO(3). The benefits of using a dataset with icosahedral rotation lies in the symmetry between taking a single-point orbit and a multi-point orbit in a vector space under the action of A_5, and the effect this has on the form of the invariant polynomials.

As detailed in the previous section, we find the irreps of A_5, then apply Gatermann's algorithms to obtain the invariant polynomials for an invariant model on the regular representation. Finally, we extend the utility of the invariant polynomials to arbitrary vector spaces with application of Weyl's theorem.

We have a comfortable outline for constructing the model, next we will discuss the dataset for an invariant model.

4.1 Finding a dataset, current implementations and issues

Many spherical datasets are free to use, however there is an inconvenient issue of arbitrarily choosing a spherical dataset: there is no spherical symmetry in the structure of the discretisation. This section closely follows the code[1] and it is encouraged to use the plotting tools provided with various parameters.

It is well known that there are no maps without distortion, in short the current high resolution spherical datasets are either cylindrical or an attempt at an "equal area" discretisation, where the points on the sphere aim to cover the sphere evenly with the drawback of few symmetrical properties to take advantage of. Datasets that exhibit more symmetry are of great use for testing invariant models against naive data augmentation models. By comparing the two on a symmetrical dataset the naive model is justly tested as it may have the capability to capture rotational invariance.

The challenge at hand is to provide a nice compromise with minimal distortion and icosahedral symmetry. The lack of datasets fitting these requirements is a huge setback for invariant modelling as it bottlenecks the previous steps and requires care to ensure the structure of each datapoint is correct for the model. Below I will outline the various options and explain the reasoning behind the choices made.

[1] https://github.com/vibabattista/Equivariant-Machine_Learning/blob/master/sphere.ipynb

4.1.1 Spherical coordinate discretisation

The first and simplest discretisation is a uniform gridspace of two angles, $\alpha \in [0, 2\pi]$ and $\beta \in [0, \pi]$, corresponding to the standard spherical coordinate system where the grid space provides $n_\alpha \times n_\beta$ points on the sphere of the form $(1, \alpha, \beta)$ in three dimensional space.

Of course when the angles in spherical coordinates are evenly spaced we have an enhanced covering of the poles as below:

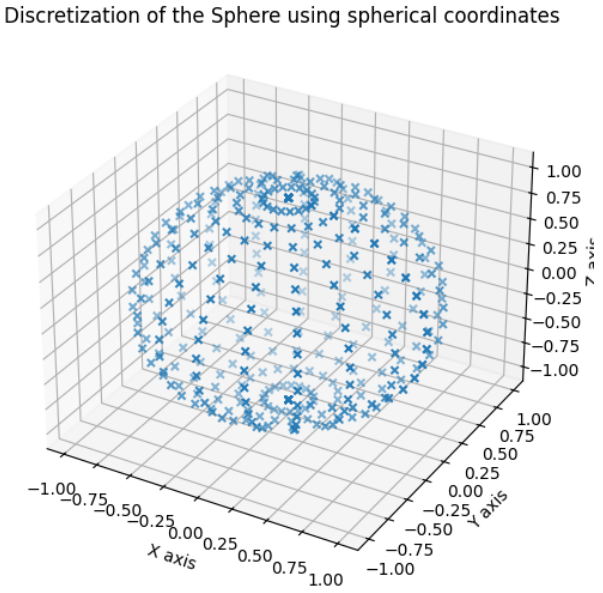

Figure 4.1 A uniform discretisation of angles. Note the heavy clusters at the poles and the sparse covering of the equator.

One somewhat popular yet lazy method is to attempt achieving rotational invariance by performing a convolution azimuthally, and potentially along lines of altitude also to achieve translation invariance in two directions. This is a lazy discretisation for the reason that any convolution in altitude will result in heavy distortion at poles (see Figure 4.2).

Despite the evident drawback of being unable to act on this discretisation with a general element of A_5, it is still provided in a convenient grid resulting in a matrix format. However, we are unable to test a model invariant to icosahedral rotations against a data augmentation model, so we should abandon convenience and move toward a more fitting discretisation for our method.

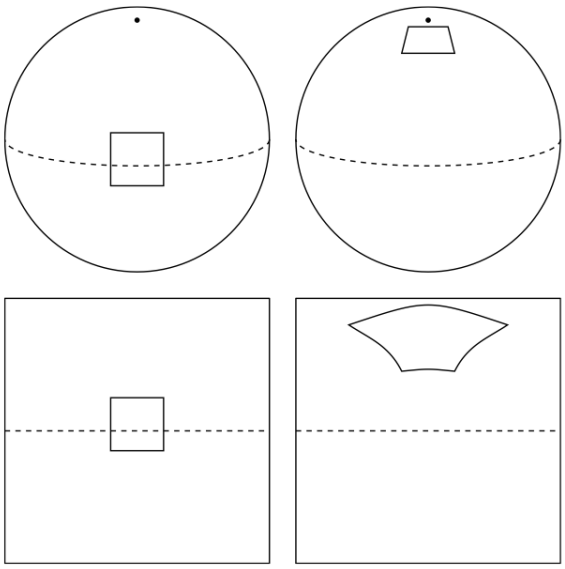

Figure 4.2 Visualisation of distortion when attempting to approximate rotation by planar translation. We want the square kernel input to correspond with equal area on the sphere, but in this case on the square matrix discretisation we would require a non-conventional distorted kernel.[2]

4.1.2 Euler-XYZ angles

To remove the distortion we aim to create a more evenly spaced distribution of points, that do not only focus on the previous two degrees of freedom (α and β). Naturally, Euler-XYZ angles (or with the same effect, Euler-ZYZ) are more appealing as three degrees of freedom is enough to describe any SO(3) rotation and consequently all icosahedral rotations.

The idea behind Euler-XYZ angles is to construct an element of SO(3) via composition of rotation about the x, y and z-axis, respectively, through three distinct angles. To understand why this is different from using spherical coordinates it is helpful to remember that while spherical coordinates can describe any point on the sphere they do not describe every possible sphere rotation. A common pitfall is to equate describing every point on the sphere to describing every rotation, recall that a spherical coordinate describes a point relative to a frame of reference initial point, and not a rotation.

A short example will make this concrete: suppose the initial point, p, on the sphere is at the north pole. Now consider the set of rotations that send p to itself, this set is isomorphic to SO(2) as the sphere can spin on the axis defined by p. Since this is not a singleton it is immediate that two points cannot describe an element of SO(3).

Euler-XYZ angles grant us three degrees of freedom to describe an element of SO(3), the rotation matrices about each axis are:[13]

$$R_x(\theta_1): \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_1) & -\sin(\theta_1) \\ 0 & \sin(\theta_1) & \cos(\theta_1) \end{bmatrix}$$

$$R_y(\theta_2): \quad \begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) \\ 0 & 1 & 0 \\ -\sin(\theta_2) & 0 & \cos(\theta_2) \end{bmatrix}$$

$$R_z(\theta_3): \quad \begin{bmatrix} \cos(\theta_3) & -\sin(\theta_3) & 0 \\ \sin(\theta_3) & \cos(\theta_3) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Defining a mesh on three different angles produces a vast array of discretisations. However including more rotations does not imply any symmetries, in fact it implies the opposite. This attempt of creating an even area covering of the sphere is far less conveniently stored in an $N_1 \times N_2 \times N_3$ array.

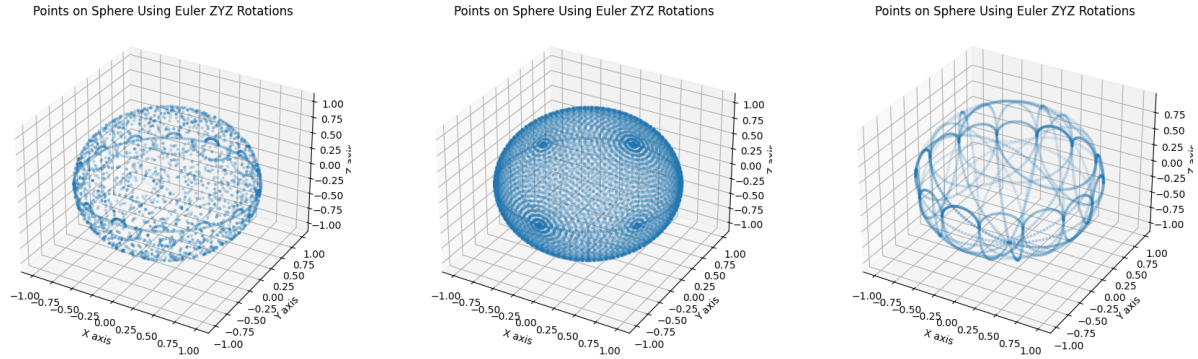

Figure 4.3 Three examples for different step sizes for each angle: (20,20,8), (100,50,4), (8,80,8), respectively.

Figure 3.3 demonstrates the variety and unpredictibility of this method, while it seems to be much closer to a uniform distribution, it does so in strange patterns that likely do not produce the uniformality we are striving for.

4.1.3 Fibonacci lattice

Achieving a near-uniform discretisation is possible with the Fibonacci lattice method.

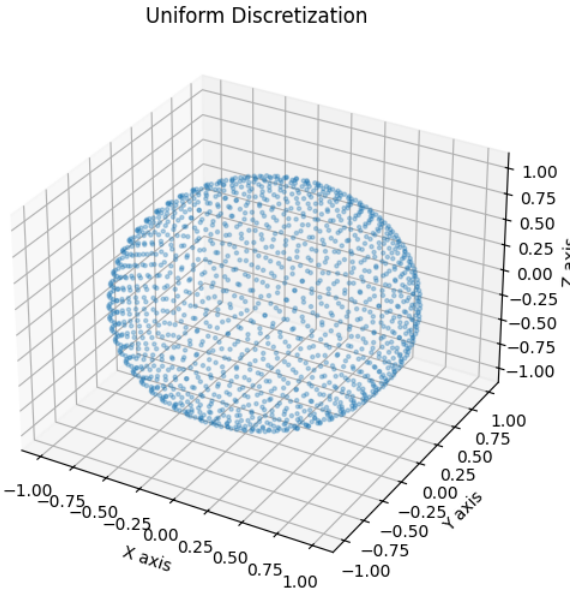

Figure 4.4 Fibonacci lattice, providing an evenly distributed discretisation.

This lattice is constructed using the golden ratio, $\phi = \frac{1+\sqrt{5}}{2}$. The choice of an irrational number is evident when considering iterative rotations on a circle by $\phi\pi$, which will never align with a previous rotation. A similar lattice is achieved on the sphere through the use of spherical coordinates. Again, it is difficult to work with this data as it cannot be expressed as a matrix, but it is a valid way to ensure a more even distribution of points.

Turning attention back to the problem at hand we see that whilst we have successfully distributed the points evenly across the sphere we have ignored the benefits of including icosahedral symmetry, so this is the discretisation method we will delve into next.

4.1.4 Icosahedral discretisation

Taking the orbit

The simplest way to discretise the sphere such that is has icosahedral symmetry is to take a unit vector in space and generate its orbit under the action of A_5. This is possible due to there existing a three-dimensional representation.

Since the icosahedral group is can be defined via rotations through different axes, given an initial point in space we can use the angle/axis transformation to obtain every point. This requires providing the axis of rotation and the angle (or the order, for example a 5 cycle can be even steps of $2\pi/5$) for each of the 60 elements of A_5.

Example: The cycle $(12345) \in A_5$ corresponds to a rotation about the axis $(1, \phi, 0)$ with angle $\theta = 2\pi/5$. So plugging in the unit axis vector and angle into this axis/angle transformation matrix provides us with a transformation corresponding to the (12345) cycle.

$$\begin{bmatrix} \cos(\theta) + x^2(1-\cos(\theta)) & xy(1-\cos(\theta)) - z\sin(\theta) & y\sin(\theta) + xz(1-\cos(\theta)) \\ z\sin(\theta) + xy(1-\cos(\theta)) & \cos(\theta) + y^2(1-\cos(\theta)) & yz(1-\cos(\theta)) - x\sin(\theta) \\ xz(1-\cos(\theta)) - y\sin(\theta) & x\sin(\theta) + yz(1-\cos(\theta)) & \cos(\theta) + z^2(1-\cos(\theta)) \end{bmatrix}$$

Remember we are working on a ball discretised by only 60 points, the connection between obtaining the discretisation and the symmetries of the group A_5 is apparent after some labeling of points from 1 to 60. It is clear that each rotation of the shape is linked with permutations of the indicies 1 to 60.

With this matrix one can readily find the generators of the group to be any 2-cycle and any 5-cycle, reducing the problem to only finding two axis/angle pairs then finding the full set of points through composition recalling the presentation of the group defined before.

This method is fairly straightforward and the SciPy python package can provide each element of A_5 in a three dimensional representation, now defining an initial unit vector and applying each rotation in the group is enough to obtain the orbit.

Figure 3.5 shows three discretisations of the sphere that have icosahedral symmetry. The first is a rotation of the vertex, resulting in 20 points. The second is the rotation of a point that does not lie on any axis of symmetry, resulting in a full 60 points. The third is an example of how we can begin to cover the sphere by including three initial points.

The code contains a 3D interactive display of these discretisations, in which, the icosahedral symmetry is significantly clearer than an image.

Now we have a steadfast method to producing any discretisation that obeys icosahedral symmetry. The simplest is to notice that our initial points may all lie in the fundamental domain of the icosahedron, that is one of the faces. This is evident by definition, as acting on a face by A_5 will map each face to each other.

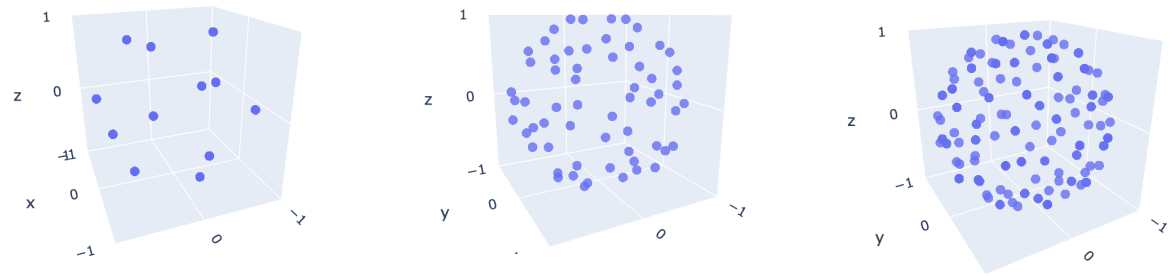

Figure 4.5 Generated icosahedron with initial point: on the vertex, off the vertex, multiple initial points, respectively

With this we have almost discretised the sphere to an arbitrarily fine grain, with very little distortion, and with the in-built symmetries we set out to achieve.

Discretising the fundamental domain

It suffices to discretise an equilateral triangle. Followed by projecting to the surface of the sphere, as the distortion is negligible and, if necessary, simple to remove.

Discretising the triangle can be done recursively, similar to the Serpinski triangle with the exception of also applying the recursion to the central triangle at each step.

Algorithm 2 shows how this can be implemented to an arbitrary depth, the depth corresponds to the number of recursions, 1 recursion being 1 point at the centre of the triangle, 2 recursions being the midpoints of the four triangles the first triangle can be divided into, then recursion on these four triangles. The number of points within the fundamental domain at depth d is therefore $\sum_{i=0}^{d-1} i^4$ which, after taking the orbit of each point, yields $60 \times \sum_{i=0}^{d-1} i^4$ unique points. By construction of the algorithm setting points to the middle of each triangle avoids points on the "edge" of the fundamental domain being repeated after taking orbits.

One input for our algorithm are the points of the fundamental domain of the icosahedron. Those points are:

$$(1, \phi, 0), (0, 1, \phi), \text{ and } (-1, \phi, 0)$$

Normalised.

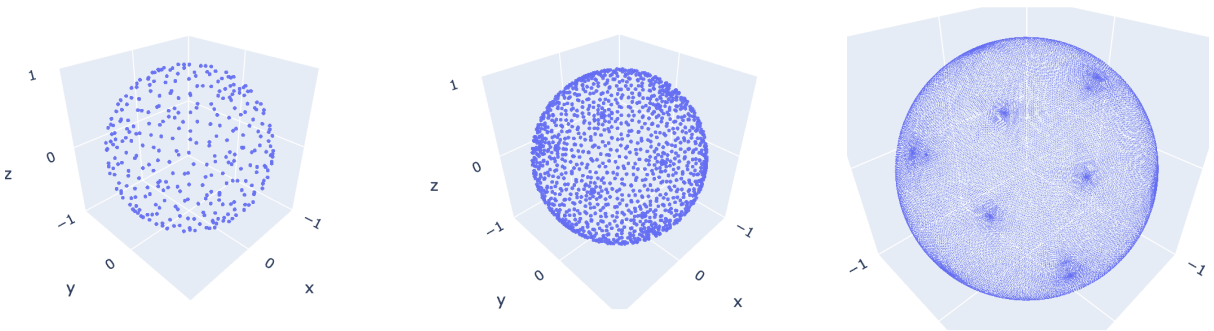

Figure 4.6 Generated mesh using algorithm 1 with recursion depth 2, 3, and 6 respectively. The third discretisation is enhanced for clarity.

Figure 3.6 illustrates how rapidly the discretisation becomes finer and more defined, as mentioned before the slight distortion from projecting a flat triangle to a sphere surface results in five pointed stars about the twelve vertices, but for the purpose of creating a dataset for an invariant model the only requirement is icosahedral symmetry, so perfectly uniform points is an unnecessary luxury that does not impact what we are testing.

Algorithm 2: Recursive function discretizing the sphere

Input: vertices, depth
Output: List of points on the sphere

Initialize: points $\leftarrow \emptyset$
Function *discretize_fundamental_domain(vertices, depth)*
  Function `FnRecursive`(*current_vertices*)
    $a, b, c \leftarrow$ current_vertices ;
    mid1 $\leftarrow \frac{a+b}{\|a+b\|}$;
    mid2 $\leftarrow \frac{b+c}{\|b+c\|}$;
    mid3 $\leftarrow \frac{c+a}{\|c+a\|}$;
    next_vertices $\leftarrow$ $[(a, \text{mid1}, \text{mid3}), (b, \text{mid1}, \text{mid2}), (c, \text{mid2}, \text{mid3}), (\text{mid1}, \text{mid2}, \text{mid3})]$;
    return *next_vertices*;
  end
  triangles $\leftarrow$ vertices ;
  for $i \leftarrow 0$ **to** *depth* **do**
    temp $\leftarrow \emptyset$;
    for *triangle* $\in$ *triangles* **do**
      temp $\leftarrow$ temp + `FnRecursive`(triangle) ;
    end
    triangles $\leftarrow$ temp ;
    print(|triangles|) ;
  end
  for *triangle* $\in$ *triangles* **do**
    $a, b, c \leftarrow$ triangle ;
    point $\leftarrow \frac{a+b+c}{\|a+b+c\|}$;
    points.append(point) ;
  end
  return *points*;
end

phi $\leftarrow \frac{\sqrt{5}+1}{2}$;
vertices $\leftarrow [(array([1, \text{phi}, 0]), array([0, 1, \text{phi}]), array([-1, \text{phi}, 0]))]$;
points $\leftarrow$ discretize_fundamental_domain(vertices, 2) ;
num_initial_points $\leftarrow$ len(points) ;

4.2 Designing the dataset

So far we have created an arbitrarily fine discretisation of the sphere which obeys icosahedral symmetry, unfortunately pre-made datasets that fit this design are scarce. In addition to this, even if we were to attempt interpolating from a dataset in, say, the spherical coordinate gridspace to the icosahedral discretisation, many frustrating issues arise due to the lack of ordering of the points introduced from both the unconventional splitting of the fundamental domain and the absence of order in the elements of A_5. Even an arbitrarily defined order would be meaningless, implying the interpolation must be done one section at a time, each section comparing a warped grip to a list of points defined via irrational angles. Needless to say, this, coupled with the warping liability in the spherical coordinates grid, should be avoided. Instead I will design a simple, toy dataset from the ground up.

The exacting nature of storing the points on the icosahedral discretisation described above should not set us back, in fact being aware of the poor ordering should ease the frustration of storing the data, making it clear that the data is best stored in an $N_{initialpoints} \times 60$ matrix. By algorithm 1 we know the rows are ordered iteratively, so we should not attempt to bring more structure to the matrix, instead we can even view each point on the fundamental domain as randomly placed, since we constructed them in this way only to have an even distribution. In addition, each column represents a rotation applied to the set of initial points.

Given this method of storage we can design a diverse set of image classes, shapes, and a many variations of each shape.

4.2.1 Shape functions

The aim of this section is to create a useful dataset of images on the sphere. As they are defined on a sphere in three dimensions and our set of points discretising the sphere has no meaningful order, we should store the three-dimesnional coordinates for each point in the discretisation.

Attempting to define shapes this way is far more natural and avoids shortcuts that may later provide issues, such as shapes that are more difficult to identify, or shapes constructed using the "ordering" of the storage matrix, in this instance the algorithm has a perfect memory of where the sections of the sphere corresponding to faces on the icosahedron are in relation to one another, however we see this as random due to the arbitrary ordering.

To prove invariance, simple distinct shapes will suffice, as not to introduce unnecessary complications. The shapes defined using polar coordinates are: "Great Band", "Checkerboard" and "Polar Caps". In each instance the points on the sphere are given 1 if they belong to the shape and 0 else, when visualised only points with 1 are plotted.

Figure 3.7 shows the points that lie inside each variation of the example shape, these are plotted from our storage matrix and in these cases each is a binary matrix of 1s and 0s.

Another type of image can be separately defined for either classification or regression problems: a shape function mapping to [0,1], which can be interpreted as a colour scale for an image.

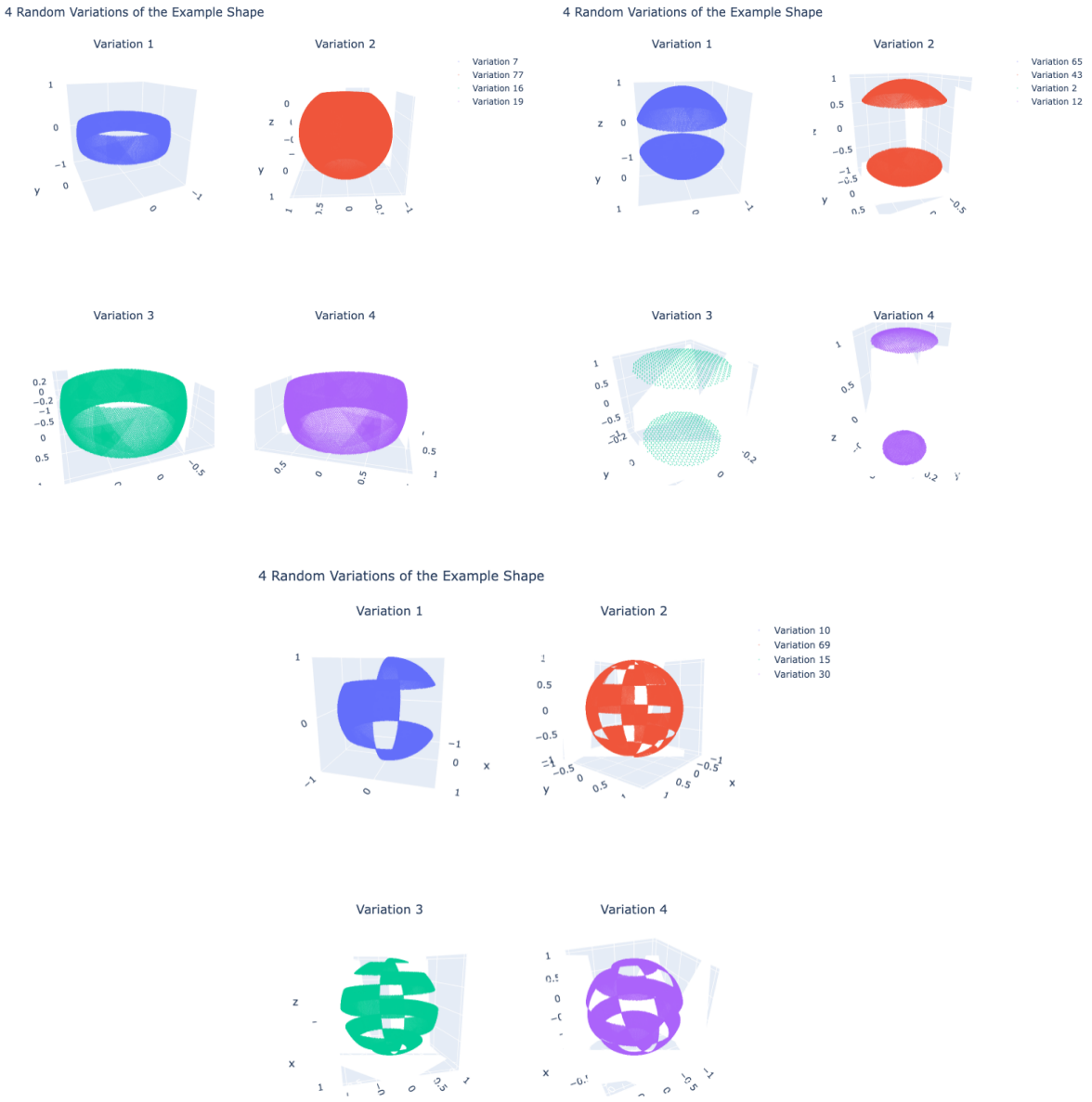

Figure 4.7 Variations of the three shape classes. Images generated randomly from code given some example shape function.

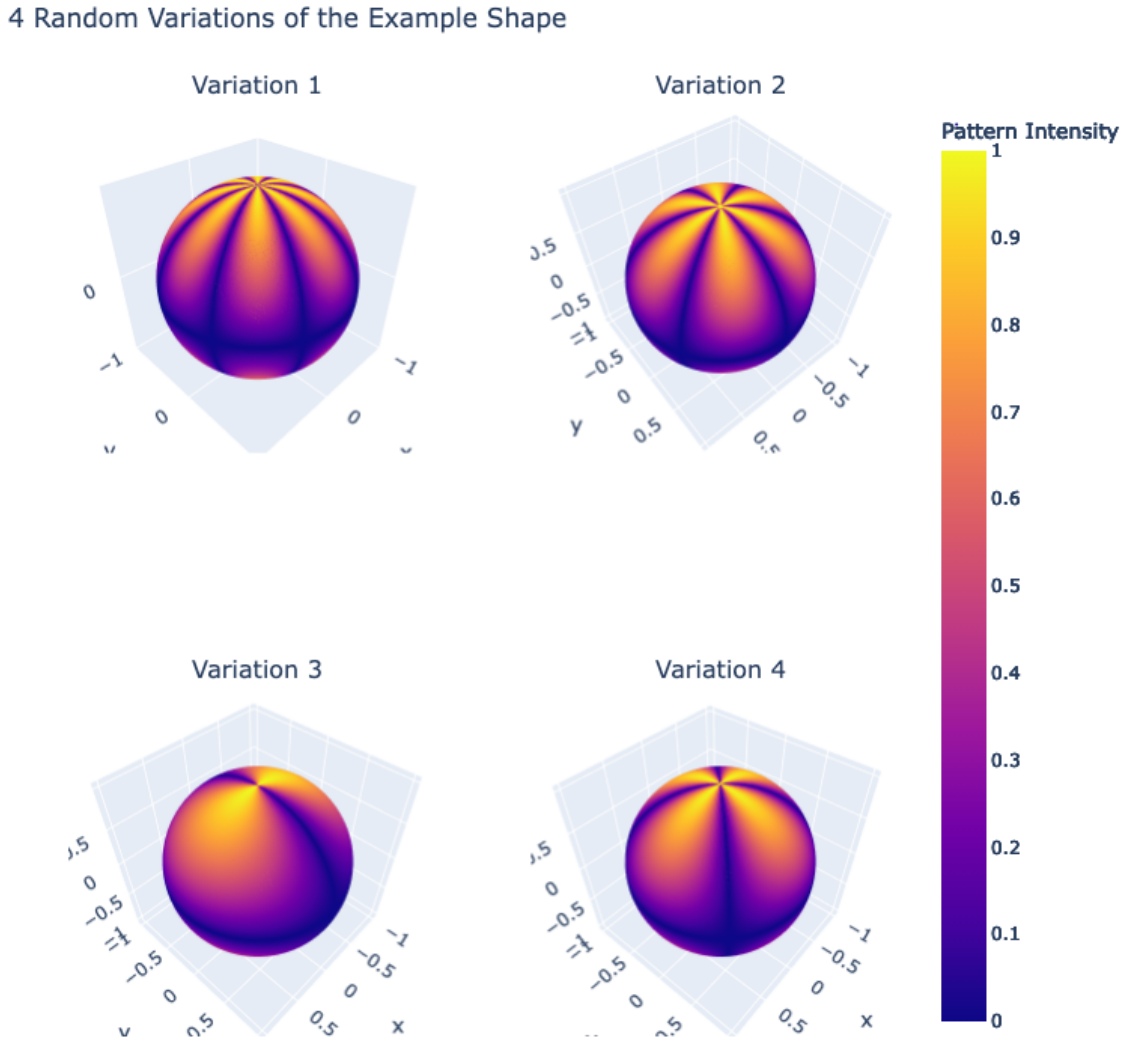

Figure 4.8 Spherical bouquet. A function class on the sphere that generates images of n-leaved Marigolds on the surface of the sphere.

Figure 3.8 illustrates the ability to design more general images. The visualisation tools are found in the code, and the functions serve as a template for constructing a diverse synthetic dataset, fitting to a range of problems. Constructing these various types of datasets can become tedious, the code linked in the appendix provides a systematic, efficient and accessible outline to customise a dataset:

- Choose a discretisation level, this dictates the number of points in the discretisation (pixel definition).
- Define shape functions. The shape functions are the most tedious step as they rely on defining a function from $\mathbb{R}^3$ to $\mathbb{R}$ to achieve the desired shape on the unit sphere. The provided shape functions take advantage of spherical coordinates to achieve this, as defining "angle" regions/rays from the origin alleviates the need to define regions in $\mathbb{R}^3$. These functions are adaptable and serve as a simple small dataset for our purposes.

- Shape parameters. Each function describes a class of shapes, within each shape class we require diversity. For example changing the number of divisions on the spherical chessboard above provides a very diverse array of spherical chessboards.
- The code is set up to generate your shape matrix, which in the case of the binary shapes is a binary matrix. Subsequently the code filters the coordinates in $\mathbb{R}^3$ by 1s and 0s and plots four random variations in a three dimensional interactive visualisation tool.

Considering the advancements in equivariant machine learning, the lack of "symmetrical" datasets, and the infeasibility of constructing such datasets the code has been made publicly available.

Chapter 5

Discussion

To recap, we have a parameterisation for a invariant and equivariant models, the relevant mathematical objects to build it and a synthetic dataset to train and test on. The parameterisation for an invariant function is

$$f_{inv}(x) = \sum_{t=1}^{T} f_t(\pi(\mathbf{A}_t x))$$

with learnable linear transformations, A_t, learnable continuous functions, f_t, and predetermined invariant polynomials π.

5.1 Next steps

With this groundwork firmly in place one can expand on invariant and equivariant machine learning through more standard machine learning test, such as the following:

- Test a simple invariant model against the data augmentation method. The datasets template provided contains icosahedral symmetries such that it is suitable for data augmentation through rotation. This comparison is achieved by training a standard neural network on augmented data and training the invariant model on the original data and comparing metrics such as time taken, accuracy and generalisation error. The key difference between the invariant model and a standard neural network trained on rotated data is the invariant model is designed to be truly invariant, whereas the data augmentation method will attempt to learn the invariance. Based on this we can expect the augmentation method to be far more computationally expensive, as the size of the dataset will increase by a factor of the group order. What is left is to compare the accuracies of these methods for models with a similar number of parameters.
- Compare models built with different T values. The idea behind making A_t learnable is such that the model puts more emphasis on the most valuable relations between the points. A reasonable prediction is increasing T yields decreasing efficiency with respect to number of parameters, in essence, T is a hyperparameter to be tuned.
- For specific problems, such as estimating invariant functions for non-finite groups, the invariant form can be built upon with other methods such as convolution or data augmentation. A non-finite group mentioned prior is SO(3), and the idea here is to capture SO(3) invariance by introducing an icosahedral rotation invariant model and infusing it with convolution. This idea is similar to that of standard convolution on a sphere, with much less impactful distortion.

- Analysis of how well the model can learn. Learning a specific form of model inherently restricts the function space of learnable functions, if it transpires that these models struggle to converge then various regularisation methods should be considered to enrich the function space and restrict it during training.

The fusion of machine learning and invariant theory represents a uniquely powerful concept in mathematics and computer science, offering substantial opportunity for modelling systems characterised by symmetry. This paper explored the application of invariant theory to systems with defined symmetries and provided a structured approach to model and study them. Another equally persuasive aspect of the field is to approach a system with unknown symmetry and construct a model that can learn it- an area that greatly complements the study of known symmetries. Models that succeed in both identifying and learning symmetrical systems together will have entirely harnessed symmetry through group theory. It is not unreasonable to suggest a family of models that can recognise, discern, and exploit symmetries that have previously gone unnoticed, the implications of which are overwhelmingly innovative.

Appendix A

First Appendix

A.1 Github code

All the code referenced in this thesis can be found here: https://github.com/vibabattista/Equivariant-Machine_Learning/tree/master

A.2 Alternative methods for finding irreps

There are various computer packages such as "escnn" and "irreps" claim to be able to provide the irreps of the icosahedral group, and various other groups with convenient functions. However it is only worth installing these on a Linux system. Potential workarounds that were attempted include: editing the meson.build files to be compatible with the macOS system, and recreating the algorithms from the source code. Both had little success as the source code was heavily dependent on the package py3nj, the root of the issue.

For those who wish to find the irreps quickly I suggest using Ubuntu Linux to use escnn, or to at least study the source code if finding other group properties or irreps for other groups is desirable.

Another method worth studying is from "Irreducible Characters of the Icosahedral Group"[14]. Code exists on the github following the methods in this paper. The code includes obtaining eigenvectors for a representation of the conjugacy classes from their interactions, matching eigenvalues and more. The paper contains an unnecessary depth of theory for the contents of this thesis to obtain the irreps. Nonetheless it is an excellent guide for understanding the irreducible character table for A_5.

A.3 List of irreps

List of all irreps of all elements provided in SymPy matrix format.

One-dimensional irrep

Three-dimensional irrep 1

Three-dimensional irrep 2

Four-dimensional irrep

Five-dimensional irrep

Bibliography

[1] K. Gatermann. *Computer algebra methods for equivariant dynamical systems*. Springer, 2007.

[2] T. Cohen and M. Welling. Group equivariant convolutional networks. In *International conference on machine learning*, pages 2990–2999. PMLR, 2016.

[3] M. Finzi, M. Welling, and A. G. Wilson. A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups. In *International conference on machine learning*, pages 3318–3328. PMLR, 2021.

[4] W. Fulton and J. Harris. *Representation theory: a first course*, volume 129. Springer Science & Business Media, 2013.

[5] G. M. Haight. The weierstrass approximation theorem. 1967.

[6] M. Golubitsky, I. Stewart, and D. G. Schaeffer. *Singularities and Groups in Bifurcation Theory: Volume II*, volume 69. Springer Science & Business Media, 2012.

[7] D. Yarotsky. Universal approximations of invariant maps by neural networks. *Constructive Approximation*, 55(1):407–474, 2022.

[8] K. Hornik, M. Stinchcombe, and H. White. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural networks*, 3(5): 551–560, 1990.

[9] M. Wesslén. The irreducible representations of a5, 2005.

[10] W. M. Hu, J. L. Yong, J. Zhou, and F. Shu. The irreducible representation matrices of the icosahedral point groups i and ih. *Superlattices and microstructures*, 3(4):391–398, 1987.

[11] K. Shirai. The basis functions and the matrix representations of the single and double icosahedral point group. *Journal of the Physical Society of Japan*, 61(8):2735–2747, 1992.

[12] L. L. Everett and A. J. Stuart. Icosahedral (a 5) family symmetry and the golden ratio prediction for solar neutrino mixing. *Physical Review D*, 79(8):085005, 2009.

[13] Wikipedia contributors. Rotation matrix — Wikipedia, the free encyclopedia, 2024. [Online; accessed 7-June-2024].

[14] S. Kanemitsu, J. Mehta, and Y. Sun. Irreducible characters of the icosahedral group. *: , , ,* 14(4):405–412, 2023.