TENSOR ISOMORPHISMS: NOTES FROM THE SHONAN MEETING - DAY 2

MAKSYM ZUBKOV

Contents

Date: June 4, 2026.

1. Speaker 1: The Missing Structure in Tensors

James B. Wilson, Colorado State University
On the missing structure in tensors

James B. Wilson works on computational algebra, isomorphism problems, tensors, and related algebraic structures. This talk was a tutorial-style reflection on what it means to recognize, compare, and classify tensors. The main theme was not only how to decide whether two tensors are isomorphic, but also what kind of evidence should count as meaningful "structure" when we say that two tensors are not isomorphic.

The talk was organized as eight reflections on a problem. The goal was to share ideas and experience about tensors, isomorphism, invariants, and the difficulty of finding intrinsic structure.

1.1. **What is a structure?** Let us start with the first important question:

What is a "structure"?

The talk began with examples. We were shown three graphs, two of which were isomorphic and one of which was not. Then we considered two groups:

$$D_8 = \langle a, b \mid a^4 = 1,\ b^2 = 1,\ bab^{-1} = a^{-1} \rangle$$

and

$$Q_8 = \{\pm 1, \pm i, \pm j, \pm k\},$$

the quaternion group.

Are these groups isomorphic?

To answer such questions, we usually use invariants. But this immediately raises another question: what kind of invariants actually explain the structure, and what kind of invariants only move the problem somewhere else?

1.2. **Leibniz's law and structuralism.** One starting point is Leibniz's law:

$$x = y \quad \Longleftrightarrow \quad \forall P,\ \big(P(x) \leftrightarrow P(y)\big).$$

That is, two objects are equal if they satisfy the same properties.

For isomorphism, one replaces equality by structural sameness:

$$x \cong y \quad \Longleftrightarrow \quad \forall P \in \mathcal{C},\ \big(P(x) \leftrightarrow P(y)\big),$$

where $\mathcal{C}$ is the class of properties allowed in the relevant language.

Examples:

- for graphs, the basic structure is the edge relation;
- for algebras, the basic structure is given by equations;
- for sets, the basic structure is membership.

In axiomatic set theory, for example, extensionality says:

$$A = B \quad \Longleftrightarrow \quad \forall x,\ (x \in A \leftrightarrow x \in B).$$

1.3. **An early issue: haecceity.** An early issue is the principle of haecceity: the question of what makes an individual object this object rather than another one.

For example, over the complex numbers, the equation

$$x^2 + 1 = 0$$

has two solutions. Algebraically, all we can say is that there are two roots. But an automorphism of the field can swap them.

So structuralism can say:

> There are two roots.

But structuralism cannot intrinsically say:

> This one is i and that one is $-i$.

To distinguish the two roots, we need something external: a convention, an embedding, an orientation, or some additional structure.

This is already a warning for invariants. Sometimes invariants do not identify objects intrinsically; they only identify them relative to another structure.

1.4. **Other concerns about structure.** Other examples of "structure" also create difficulties.

For instance:

- one knot is not equivalent to another because their fundamental groups are different;
- two groups are not isomorphic because they have different character tables.

But these explanations shift the question to a new isomorphism problem:

- How do we know the fundamental groups are different?
- How do we compare character tables?
- What conventions were used to compute these invariants?

Thus, invariants may not solve the problem completely. They may only move the problem into a different category.

> My conclusion: philosophy of mathematics has lessons for what we are doing. Sometimes we are changing the road instead of changing the bike.

Invariants like these often only shift the question to the next isomorphism problem.

There was also a brief mention of argumentation theory and erotetic logic, the logic of questions. This is relevant because isomorphism testing is not just about producing answers; it is also about asking the right questions.

1.5. **Where can I precisely write my isomorphism problems?** A practical issue is that computer algebra systems such as OSCAR and Magma can compute invariants. But two systems may use different conventions, different normal forms, or different choices of defining polynomials, such as Conway polynomials or Lübeck polynomials.

So even if two computations are mathematically equivalent, they may not be literally identical.

This leads to the question:

Where can I precisely write down my isomorphism problem?

Perhaps the right language is categorical.

In a category, one does not only say whether

$$A \cong B.$$

One also records the data that counts as an isomorphism.

So instead of thinking only about invariants, we can think about the category or groupoid in which the objects live.

1.6. **Categorical and groupoid models.** In a categorical or groupoid model,

$$A \cong B$$

means that there is specified data which counts as an isomorphism from A to B.

The usual laws of isomorphism correspond to groupoid structure:

- reflexivity gives identity morphisms;
- symmetry gives inverses;
- transitivity gives composition.

This leads toward ∞-groupoid viewpoints, such as the Hofmann–Streicher groupoid model from the 1990s, where equality and isomorphism are treated recursively and at higher order.

For example, consider the left-handed and right-handed trefoil knots. We may want to distinguish them. Unfortunately, their fundamental groups are isomorphic, so this invariant alone does not see the difference.

Thus, even a powerful invariant may forget the structure we actually care about.

My second conclusion: it is hard to outdo a century of abstraction.

1.7. **How many n-dimensional associative algebras are there?** Now consider associative algebras.

Let A be a k-vector space with a distributive product

$$* : A \times A \to A$$

satisfying associativity:

$$x * (y * z) = (x * y) * z.$$

A natural classification question is:

How many n-dimensional associative algebras are there?

Some structural facts are known.

- By Wedderburn theory, simple finite-dimensional algebras are matrix algebras over division algebras.
- By the Jacobson radical theory, an algebra can often be studied through its semisimple quotient and its nilpotent radical:
$$A/\operatorname{rad}(A) \cong \bigoplus_i S_i,$$
where the S_i are simple.
- Nilpotent structure can be studied using filtrations, such as powers of the radical.

Groups can also be studied through related algebraic structures, but they may behave badly from the point of view of classification. Filtrations help organize the complexity.

Conclusion: when you do isomorphism, you do counting. We have

$$\text{counting} \cong \text{comparing} \cong \text{classifying}.$$

Counting objects, comparing objects, and classifying objects are deeply related tasks.

1.8. **Can algebraic structure be useful?** The next question was:

Can I actually be useful by doing algebraic structure?

One motivation comes from clustering. We may have labeled data, and we want to discover structure in it.

The discussion then moved toward entropy and tensor constructions. The idea is that by permuting or swapping rows and columns, one may reveal hidden tensor structure.

Homomorphisms of products should have analogues of ideals. Entropic ideals and entropic varieties give one possible way to understand such structures.

Conclusion: tensors make sense for all entropic systems.

This includes:

- scalars;
- modules;
- tensor products;
- hom–tensor adjunctions;
- algebraic systems where operations interact through entropic laws.

1.9. **Issues with Whitney's tensor product.** The talk then discussed issues with Whitney's tensor product.

Some concerns were:

- it is not defined for all modules in the desired generality;
- it is not associative in the expected way;
- it only captures nearest-neighbor behavior;
- the lemmas needed for computation are cryptic;
- it lacks a categorical definition.

Conclusion: you can ask your tensor what tensor product it wants.

In other words, rather than forcing all tensors into one predetermined tensor product, one can try to infer the right tensor product structure from the tensor itself.

1.10. **Will Lie tensor products help anyone?** The next question was whether Lie tensor products can be useful.

One possible application is to continuous clustering algorithms, where the algebraic structure gives a way to organize and compare data continuously.

This connects with Wilson's broader program of using algebraic operators and tensor structure to detect patterns in tensor data.

1.11. **What about isomorphism?** Finally, the talk returned to the original question:

What about $\cong$?

One important theme is that isomorphism should not only be a yes/no answer. We also want reproducible evidence.

If two tensors are isomorphic, an explicit isomorphism is easy to verify.

But if two tensors are declared non-isomorphic, the evidence is often much harder to check. It may depend on:

- complicated invariants;
- machine computations;
- implicit conventions;
- choices of coordinates;
- choices of normal forms;
- choices made by different software systems.

This makes tensor non-isomorphism difficult to communicate and reproduce.

The philosophical concern is that tensors may not have a single missing piece of structure waiting to be discovered. Instead, different contexts may demand different kinds of structure.

1.12. **Takeaway.** The main takeaway is that tensor isomorphism is not only an algorithmic problem. It is also a structural and philosophical problem.

We want to know:

- What counts as structure?
- What counts as an invariant?
- What counts as reproducible evidence for non-isomorphism?
- In what category or groupoid should the isomorphism problem live?
- Are we discovering intrinsic structure, or only imposing external conventions?

For my own interests, the most relevant point is that invariants do not automatically solve isomorphism problems. They often create new isomorphism problems at the next level.

This is important for tensor isomorphism, group isomorphism, and also for algebraic geometry of neural networks: when we compare two algebraic or tensorial objects, we should be explicit about the ambient category, the allowed transformations, and the meaning of the invariants we compute.

2. Speaker 2: Computing Normalisers of Permutation Groups

Colva M. Roney-Dougal, University of St Andrews

Computing normalisers of permutation groups

Colva M. Roney-Dougal works in computational group theory, especially permutation groups and algorithms for finite groups. This talk was about the normaliser problem for permutation groups and its relationship to isomorphism problems.

The main theme was that normalisers are not only a natural object in group theory, but also a useful computational tool: if an element normalises a subgroup, then it induces an automorphism of that subgroup. This creates a direct connection between normaliser computation, conjugacy, automorphism groups, and isomorphism testing.

2.1. **Introducing normalisers.** The normaliser problem is the following.

Given two subgroups

$$G, H \leq S_n,$$

compute

$$N_G(H) \coloneqq \{g \in G \mid g^{-1}Hg = H\}.$$

Equivalently,

$$N_G(H) = \{g \in G \mid g^{-1}hg \in H,\ \forall h \in H\}.$$

Here G and H are given by generating sets. We assume that the generating sets have size at most n, and we measure complexity as a function of n.

A special case is when

$$G = S_n.$$

This is called the symmetric normaliser problem, or NormSym.

2.2. **The conjugacy problem.** A closely related problem is the conjugacy problem.

Given

$$G, H, K \leq S_n,$$

decide whether there exists $g \in G$ such that

$$H^g = K,$$

where

$$H^g = g^{-1}Hg.$$

The special case where

$$G = S_n$$

is called the symmetric conjugacy problem, or ConjSym.

Theorem 2.1 (Luks, 1993)**.** *The normaliser problem and the conjugacy problem are polynomial-time equivalent.*

So computing normalisers and testing conjugacy of subgroups are essentially the same problem from the point of view of polynomial-time reductions.

It is not known whether NormSym is as hard as the general normaliser problem.

The best known general bound for NormSym is simply exponential:

$$2^{O(n)}$$

due to Wiebking.

2.3. **The Luks hierarchy.** The talk then placed the normaliser problem in the broader landscape of computational group theory.

Theorem 2.2. *Group isomorphism reduces to the intersection problem: given $G, H \leq S_n$, compute*

$$G \cap H.$$

The intersection problem is polynomial-time equivalent to several other problems, including:

- *set stabiliser;*
- *element centraliser;*
- *other problems in the Luks class.*

Moreover, the intersection problem reduces to the normaliser problem.

Thus, the normaliser problem sits high in the Luks hierarchy of permutation group problems.

Theorem 2.3. *Problems in the Luks class can be solved in quasipolynomial time.*

This is one reason normaliser computation is important: it is closely connected to the algorithmic complexity of isomorphism problems.

2.4. **Some background from permutation groups.** The next part of the talk reviewed several standard methods from permutation group algorithms.

The main goal is to reduce the number of elements of G that we need to consider.

- The centraliser
$$C_G(H) \coloneqq \{x \in G \mid x^{-1}hx = h,\ \forall h \in H\}$$
consists of elements of G that commute with every element of H.

  For $H = \langle X \rangle$, one can compute centralisers efficiently, for example in time roughly
$$O(|X|n^2)$$
in suitable permutation group settings.
- If $H^g = H$, then conjugation by g induces an automorphism of H. Therefore,
$$N_G(H)/C_G(H)$$
embeds into
$$\mathrm{Aut}(H).$$
This is one of the key links between normalisers and automorphism groups.
- An orbit of H is a set of the form
$$\alpha^H = \{\alpha^h \mid h \in H\}.$$
The normaliser
$$N = N_G(H)$$
permutes the orbits of H.
- An orbital of H is an orbit of H on ordered pairs:
$$\{(\alpha^h, \beta^h) \mid h \in H\}.$$
Orbitals provide more refined information than ordinary orbits.

These structures help restrict the possible normalising elements.

2.5. **Transitive and primitive groups.** A permutation group $G \leq S_n$ is transitive if it has only one orbit on
$$\{1, \dots, n\}.$$
It is primitive if it is transitive and preserves no nontrivial block system.

Primitive groups are the building blocks of transitive permutation groups. More precisely, every transitive group can be embedded into an iterated wreath product of primitive groups.

This is important because algorithms for permutation groups often proceed by reducing to primitive components.

Problems in the Luks class are generally easiest when the group has many orbits. Many orbits give more combinatorial structure and more places where the algorithm can split the problem.

2.6. **Groups of affine type.** The next discussion was about groups of affine type.

A primitive group of affine type has the form
$$H = V \rtimes K,$$
where
$$V \cong \mathbb{F}_p^d$$
acts regularly by translations, and
$$K \leq GL_d(p)$$
acts linearly.

In this setting, primitivity of H is equivalent to irreducibility of the linear group K on V.

So the permutation group question becomes related to a linear algebra question:

$$K \leq GL_d(p) \quad \text{irreducible?}$$

This is a common theme in computational group theory: permutation group problems are often reduced to linear group problems.

2.7. **Bases for permutation groups.** A base for a permutation group

$$G \leq S_n$$

is a sequence of points

$$(\beta_1, \ldots, \beta_b)$$

from

$$\{1, 2, \ldots, n\}$$

such that the pointwise stabiliser is trivial:

$$G_{\beta_1,\ldots,\beta_b} = 1.$$

In other words, the only element of G fixing all the β_i is the identity.

Bases are fundamental in permutation group algorithms, because once we know the images of the base points, we know the group element.

By Sims' algorithm, one can compute a base and a strong generating set for any permutation group in polynomial time.

Let

$$b(G)$$

denote the size of the smallest base for G.

Computing a base of minimum possible size $b(G)$ is NP-hard.

2.8. **Two invariants of primitive groups.** The talk then introduced two useful invariants of primitive permutation groups.

First, write

$$d(H)$$

for the smallest number of elements needed to generate H.

Second, write

$$b(H)$$

for the base size of H, namely the smallest size of a base for H.

These two invariants are useful because they control different aspects of the search problem:

- $d(H)$ measures how complicated H is as an abstract group;
- $b(H)$ measures how much information is needed to determine a permutation in H.

In many algorithms, small base size leads to better search procedures.

2.9. **Connection to normalisers.** If $g \in N_G(H)$, then conjugation by g gives an automorphism of H:

$$h \mapsto g^{-1}hg.$$

Therefore, normaliser computation can be viewed as the problem of finding which automorphisms of H are realised by conjugation inside G.

This gives the quotient embedding

$$N_G(H)/C_G(H) \hookrightarrow \mathrm{Aut}(H).$$

Thus, one possible strategy is:

- compute or restrict possible automorphisms of H;
- determine which of them are induced by elements of G;

- reconstruct the corresponding normaliser.

This is why normaliser computation is closely related to isomorphism and automorphism problems.

2.10. **Takeaway.** The main takeaway is that normalisers of permutation groups are a central computational object.

They connect:

- subgroup conjugacy;
- automorphism groups;
- group isomorphism;
- intersection and stabiliser problems;
- the Luks hierarchy;
- primitive group theory;
- bases and strong generating sets.

For the broader workshop, the relevant point is that normaliser computation is another example of an isomorphism-type problem. It asks not only whether two objects are equivalent, but also which symmetries preserve a given structure.

For tensor isomorphism, this is a useful analogy: understanding stabilisers, normalisers, and induced automorphisms may help organize the search for isomorphisms and the computation of invariants.

3. Speaker 3: The Isomorphism Problem for Finite p-Groups

Bettina Eick, TU Braunschweig

On the isomorphism problem for finite p-groups

Bettina Eick works in computational group theory, with a focus on algorithms for groups, associative algebras, Lie algebras, and classification problems. This talk reviewed practical methods for solving the isomorphism problem for finite p-groups and explained how these methods are used in the enumeration and classification of finite groups.

The isomorphism problem for finite p-groups is one of the central difficult cases of the group isomorphism problem. Even though p-groups are nilpotent and therefore have a lot of structure, there are extremely many of them, and distinguishing them efficiently is difficult.

3.1. **The isomorphism problem.** The basic decision problem is:

Given two finite groups G and H, decide whether

$$G \cong H.$$

There are several related versions of this problem.

- **Decision version:** decide whether
$$G \cong H.$$
- **Search version:** if $G \cong H$, determine an explicit isomorphism
$$\varphi : G \to H.$$
- **Canonical form version:** compute a canonical form $\mathrm{Can}(G)$ such that
$$G \cong H \quad \Longleftrightarrow \quad \mathrm{Can}(G) = \mathrm{Can}(H).$$

The canonical form version is the strongest one. If we can compute canonical forms efficiently, then we can solve the decision problem by comparing canonical forms.

3.2. **Why finite p-groups are hard.** A finite p-group is a group whose order is a power of a prime:

$$|G| = p^n.$$

Finite p-groups are always nilpotent, so they come with useful central series and quotient structures. However, they are also extremely numerous. As n grows, the number of groups of order p^n grows very quickly.

Thus, the difficulty is not that p-groups have no structure. The difficulty is that they have too much possible structure.

This is why the isomorphism problem for p-groups is a major testing ground for algorithms in computational group theory.

3.3. **Some history for p-groups.** The talk recalled several important developments in practical algorithms for finite p-groups.

- O'Brien introduced the main practical algorithmic framework for p-groups in the early 1990s.
- Eick and O'Brien developed enumeration methods and applied them to groups of orders

$$2^9 \quad \text{and} \quad 2^{10}.$$

- Eick, Leedham-Green, and O'Brien developed methods for computing automorphism groups of p-groups.
- Besche, Eick, and O'Brien developed random methods and group identification methods, including the `IdGroup` functionality used in computational group systems.

These algorithms became important parts of the practical classification of small groups.

3.4. **Enumeration and isomorphism.** Enumeration and isomorphism are closely related.

If one wants to enumerate all groups of a given order, one must avoid counting the same group multiple times. Thus, at each step, one needs an isomorphism test.

Conversely, if one has a good classification or enumeration method, then one often obtains useful invariants or canonical representatives, which can help solve the isomorphism problem.

So the three tasks are intertwined:

$$\text{enumeration} \quad \leftrightarrow \quad \text{classification} \quad \leftrightarrow \quad \text{isomorphism testing}.$$

3.5. **Burnside's lemma.** One basic counting tool is Burnside's lemma.

Let a finite group G act on a finite set X. Then the number of orbits of this action is

$$\frac{1}{|G|} \sum_{g \in G} |\operatorname{Fix}_g(X)|,$$

where

$$\operatorname{Fix}_g(X) = \{x \in X \mid g \cdot x = x\}.$$

This is useful because many classification problems can be formulated as orbit problems.

For example, one may have a set X of possible algebraic structures, and a group G acting by change of basis. Then isomorphism classes correspond to orbits of this action.

Thus, Burnside's lemma turns an isomorphism classification problem into a counting problem involving fixed points.

3.6. **Automorphism groups.** Automorphism groups play a central role.

For a group G, the automorphism group is

$$\operatorname{Aut}(G) = \{\varphi : G \to G \mid \varphi \text{ is an isomorphism}\}.$$

Computing automorphism groups is important because automorphisms control how many equivalent choices appear during enumeration.

If one constructs larger p-groups from smaller ones, then the automorphism group of the smaller group acts on the possible extensions or descendants. To avoid duplicates, one has to compute orbits under this automorphism group.

Thus, automorphism computation is not a side problem. It is one of the main ingredients in practical p-group enumeration.

3.7. **The p-group generation philosophy.** The practical approach is often not to list all multiplication tables directly. Instead, one builds larger p-groups from smaller ones.

Very roughly, one starts with a smaller quotient and constructs possible descendants. Then one has to decide which descendants are isomorphic.

This leads to the following general strategy:

- construct candidates using group-theoretic structure;
- use automorphism groups to reduce the number of candidates;
- identify orbit representatives;
- test isomorphism only among the remaining candidates.

This is why the isomorphism problem, automorphism groups, and enumeration algorithms are all tied together.

3.8. **Canonical forms.** Another possible approach is to compute a canonical form for a group.

A canonical form is a representative depending only on the isomorphism class. If two groups are isomorphic, they should have the same canonical form; if they are not isomorphic, the canonical forms should differ.

In practice, canonical forms can be difficult to compute. But they are very useful because they turn isomorphism testing into direct comparison.

For p-groups, canonical forms may use structural data such as:

- central series;
- lower exponent-p central series;
- Frattini quotients;
- commutator structure;
- power structure;
- automorphism group actions.

3.9. **Connection to tensor isomorphism.** Finite p-groups, especially groups of class 2 and exponent p, are closely related to bilinear maps.

For such a group, the commutator map has the form

$$G/Z(G) \times G/Z(G) \to [G, G].$$

This is an alternating bilinear map, and therefore can be encoded as a tensor.

Thus, for this class of p-groups, group isomorphism can be translated into an isomorphism problem for bilinear maps or tensors.

This is one of the reasons finite p-groups are so relevant to a workshop on tensor isomorphism.

3.10. **Takeaway.** The main takeaway is that the isomorphism problem for finite p-groups is both very classical and very computational.

It connects:

- group isomorphism;
- group enumeration;
- automorphism groups;

- canonical forms;
- orbit problems;
- Burnside's lemma;
- tensor and bilinear-map isomorphism.

For the broader workshop, the most important point is that practical isomorphism algorithms often depend on understanding automorphism groups and orbit structures. This is also true for tensor isomorphism: to classify objects, we need not only invariants, but also effective ways to organize group actions and avoid duplicate representatives.

4. Speaker 4: Tensor Invariants and Poles of Local Zeta Functions

Christopher Voll, Bielefeld University
Tensor invariants and poles of local zeta functions

Christopher Voll works in asymptotic group theory, especially on zeta functions associated with groups, rings, and related algebraic structures. The main theme of the talk was that local zeta functions, and especially their pole spectra, can be viewed as invariants of tensors.

The guiding idea is that tensors encode algebraic structures such as commutator maps of groups or Lie rings. Zeta functions then give an analytic way to study counting problems attached to these structures, for example counting subgroups, subrings, or subalgebras of finite index.

The slogan of the talk was:

> Pole spectra of local zeta functions can be viewed as tensor invariants.

Moreover, these pole spectra may measure a kind of singularity complexity of the tensor.

4.1. **Groups from tensors.** Given

$$a, b \in \mathbb{N},$$

consider an alternating bilinear map

$$T : \mathbb{Z}^a \times \mathbb{Z}^a \to \mathbb{Z}^b.$$

From this tensor, we obtain a torsion-free class-2 nilpotent group

$$G_T = (\mathbb{Z}^{a+b}, *),$$

where the product is defined by

$$(x, y) * (x', y') = (x + x',\, y + y' + T(x, x')).$$

Here

$$x, x' \in \mathbb{Z}^a, \qquad y, y' \in \mathbb{Z}^b.$$

The tensor T controls the commutator structure of the group. Thus, changing the tensor changes the associated group.

This is one of the basic bridges between tensor isomorphism and group theory: tensors can be used to encode nilpotent groups, especially groups of nilpotency class 2.

4.2. **Zeta functions from groups.** Let G be a finitely generated nilpotent group, such as G_T.

Definition 4.1. The subgroup zeta function of G is

$$\zeta_G(s) = \sum_{H \leq_f G} |G : H|^{-s},$$

where the sum is over all finite-index subgroups $H \leq G$.

This is a Dirichlet generating function encoding the sequence

$$a_n(G) = \#\{H \leq G \mid |G : H| = n\}.$$

Equivalently,

$$\zeta_G(s) = \sum_{n=1}^{\infty} a_n(G) n^{-s}.$$

For finitely generated nilpotent groups, these zeta functions admit an Euler product:

$$\zeta_G(s) = \prod_{p \text{ prime}} \zeta_{G,p}(s),$$

where the local factor $\zeta_{G,p}(s)$ counts subgroups of p-power index.

A fundamental fact is that the local factors are rational functions in p^{-s}. More precisely, there exist integers

$$a_1, \ldots, a_N, \qquad b_1, \ldots, b_N$$

such that, for almost all primes p,

$$\zeta_{G,p}(s) = \frac{\text{complicated numerator depending on } p}{\prod_{i=1}^{N}(1 - p^{a_i - b_i s})}.$$

The possible poles therefore occur at rational numbers of the form

$$\frac{a_i}{b_i},$$

up to the usual periodicity coming from the complex logarithm.

4.3. **Local pole spectrum.**

Definition 4.2. The local pole spectrum of G, denoted

$$\mathrm{LPS}_G,$$

is the set of rational real parts of poles occurring in the local zeta functions

$$\zeta_{G,p}(s).$$

Equivalently, from a rational expression of the form

$$\zeta_{G,p}(s) = \frac{P_p(p^{-s})}{\prod_i (1 - p^{a_i - b_i s})},$$

the candidate local pole spectrum is given by the numbers

$$\frac{a_i}{b_i},$$

after accounting for cancellations with the numerator.

The point is that the local pole spectrum is much smaller than the whole zeta function, but it still contains significant asymptotic and geometric information.

4.4. **Example: the Heisenberg group.** Consider the case

$$a = 2, \qquad b = 1.$$

Let

$$T : \mathbb{Z}^2 \times \mathbb{Z}^2 \to \mathbb{Z}$$

be the standard symplectic form.

The associated group G_T is the integral Heisenberg group:

$$H = \begin{bmatrix} 1 & \mathbb{Z} & \mathbb{Z} \\ 0 & 1 & \mathbb{Z} \\ 0 & 0 & 1 \end{bmatrix}.$$

In this case, one obtains

$$\mathrm{LPS}_H = \left\{0, 1, \frac{2}{2}, \frac{3}{2}\right\} = \left\{0, 1, \frac{3}{2}\right\}.$$

So even for the simplest non-abelian nilpotent group, the pole spectrum already contains nontrivial rational data.

4.5. **Pole spectra from tensors.** Recall the general picture:

(1) A tensor T defines a nilpotent group or Lie ring.
(2) The group or Lie ring has a zeta function.
(3) The zeta function decomposes into local factors.
(4) The poles of the local factors give a local pole spectrum.
(5) This pole spectrum can be viewed as an invariant of the original tensor.

Thus, instead of studying a tensor directly, we study the analytic behavior of a zeta function attached to it.

4.6. **Pole spectra of higher Heisenberg groups.** The talk then discussed higher Heisenberg groups.

These are natural generalizations of the ordinary Heisenberg group. Their Lie lattices have generators

$$x_1, \ldots, x_{2n}, y$$

with brackets

$$[x_{2i-1}, x_{2i}] = y \qquad \text{for } i = 1, \ldots, n,$$

and all other brackets zero unless forced by skew-symmetry.

Recent work of Jianhao Shen and Christopher Voll gives explicit formulae for the subalgebra zeta functions of all higher Heisenberg Lie algebras over compact discrete valuation rings. The paper develops Hecke-theoretic methods for counting sublattices in symplectic lattices and studies local poles and functional equations.

Theorem 4.3 (Shen–Voll, 2026)**.** *The local subalgebra zeta functions of higher Heisenberg Lie algebras admit explicit formulae. These formulae can be used to study their local pole spectra.*

4.7. **Pole spectra of groups from tensors.** Many examples of such zeta functions have been computed.

For example, Gareth Taylor computed examples involving direct products of Heisenberg groups. One example mentioned in the talk was:

$$\mathrm{LPS}_{H \times H} = \left\{0, 1, 2, 3, \frac{4}{2}, \frac{5}{2}, \frac{5}{3}, \frac{7}{3}, \frac{8}{3}\right\}.$$

After simplification, this contains repeated values such as

$$\frac{4}{2} = 2.$$

The important point is not only the exact set, but the fact that local pole spectra can become richer as the tensor or associated group becomes more complicated.

4.8. **An analogy: Igusa's local zeta function.** Then the talk shifted to an analogy with Igusa's local zeta functions.

And now to something completely different.

Let

$$f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$$

and let p be a prime number.

For $m \in \mathbb{N}$, define

$$N_{f,p,m} = \#\left\{x \in (\mathbb{Z}/p^m\mathbb{Z})^n \mid f(x) \equiv 0 \pmod{p^m}\right\}.$$

Igusa's local zeta function

$$Z_{f,p}(s)$$

knows, or encodes, the sequence

$$(N_{f,p,m})_{m \geq 1}.$$

A key fact is that $Z_{f,p}(s)$ is also a rational function in p^{-s}. More precisely, it has a form like

$$Z_{f,p}(s) = \frac{\text{complicated numerator depending on } p}{\prod_{i=1}^{N}(1 - p^{\alpha_i - \beta_i s})}.$$

So one can again define a local pole spectrum by looking at the rational numbers

$$\frac{\alpha_i}{\beta_i}$$

appearing in the denominators, after cancellations.

Definition 4.4. The local pole spectrum of f, denoted

$$\mathrm{LPS}_f,$$

is the set of real parts of poles of the local Igusa zeta functions

$$Z_{f,p}(s).$$

4.9. **The monodromy conjecture.** The analogy with Igusa zeta functions is important because of the Monodromy Conjecture.

Very roughly, the Monodromy Conjecture says that poles of Igusa's local zeta function should detect monodromy eigenvalues of the hypersurface singularity defined by f.

In a simplified form:

$$r \in \mathrm{LPS}_f \quad \Longrightarrow \quad \exp(2\pi i r) \in \mathrm{MonEV}_f,$$

where

$$\mathrm{MonEV}_f$$

denotes the set of monodromy eigenvalues associated with f.

This is the conceptual analogy:

$$\begin{aligned} \text{polynomial } f \quad &\longrightarrow \quad \text{Igusa local zeta function} \\ \text{tensor } T \quad &\longrightarrow \quad \text{group/Lie zeta function} \end{aligned}$$

In the first row, poles are expected to reflect singularity-theoretic information.

In the second row, the hope is that poles of local zeta functions reflect structural or singularity-like complexity of the tensor.

4.10. **Question for neurovarieties.** A natural question for me is whether this analogy can help us understand the complexity of neurovarieties.

For example, consider the neurovariety corresponding to the architecture

$$\mathbf{d} = (2, 3, 2, 1).$$

Can one attach a local zeta function to this neurovariety, or to the ideal defining it, in such a way that its pole spectrum measures the singularity complexity of the neurovariety?

More concretely, one could ask:

> Can pole spectra of local zeta functions help distinguish neurovarieties, compare their singularities, or measure the complexity of their defining equations?

There is one important caveat. The classical Monodromy Conjecture is formulated for hypersurfaces, i.e. for one polynomial

$$f = 0.$$

A neurovariety is usually defined by an ideal with many generators, not necessarily by a single polynomial. Therefore, one would need either:

- a version of Igusa zeta functions for ideals;
- a choice of a distinguished polynomial invariant;
- or a group/tensor construction attached to the parametrization.

Still, the analogy is suggestive. If pole spectra measure singularity complexity for tensors, then they may also provide a new invariant for algebraic varieties arising from neural network parameterizations.

4.11. **Takeaway.** The main takeaway is that zeta functions give analytic invariants of algebraic and tensorial structures.

The chain of ideas is:

$$\text{tensor} \longrightarrow \text{nilpotent group or Lie ring} \longrightarrow \text{local zeta function} \longrightarrow \text{pole spectrum}.$$

The pole spectrum is much smaller than the full zeta function, but it may still contain deep information about the tensor.

For the broader workshop, this gives a new kind of invariant for tensor isomorphism problems. Instead of only using linear algebraic or representation-theoretic invariants, one can use analytic invariants coming from local zeta functions.

For my own work, the most interesting possible direction is:

> Can local zeta functions and their pole spectra be used to study singularities and complexity of neurovarieties?

This would connect tensor isomorphism, nilpotent groups, local zeta functions, singularity theory, and neuroalgebraic geometry.

5. Speaker 5: Enumeration via Tensors

Tobias Rossmann, University of Galway
Enumeration via tensors

Tobias Rossmann works in algebra, especially computational and asymptotic aspects of algebraic counting problems. His work focuses on zeta functions arising from the enumeration of subgroups, representations, conjugacy classes, and related algebraic structures.

The main theme of the talk was that many enumeration problems in algebra can be translated into tensor language. Once the relevant algebraic object has been replaced by a tensor, one can use tensor invariants, tensor equivalences, and algebraic geometry to study the associated generating functions.

The general philosophy was:

$$\text{algebraic object} \longrightarrow \text{tensor} \longrightarrow \text{geometric or arithmetic data} \longrightarrow \text{zeta function}.$$

In other words, we start with an algebraic object, get rid of some of the original algebraic packaging by extracting a tensor, and then build a zeta function from the tensor.

5.1. **Group functors and group schemes.** A group functor is a functor

$$G : \mathrm{CommRings} \to \mathrm{Groups}.$$

Thus, for every commutative ring R, we get a group

$$G(R).$$

A group scheme is a group functor of algebro-geometric origin. In practice, this means that the entries and multiplication rules of the group are given by polynomial equations.

Example 5.1. Let U_d denote the group of upper unitriangular $d \times d$ matrices:

$$U_d(R) = \{A \in GL_d(R) \mid A \text{ is upper triangular and has all diagonal entries } 1\}.$$

This defines a unipotent group scheme.

Unipotent groups are particularly important because they are closely related to nilpotent Lie algebras, and nilpotent structures often lead naturally to tensors.

5.2. **Local counting problems in unipotent groups.** Let

$$G \leq U_d$$

be a unipotent group scheme, and let p be a prime.

The local orbit-counting zeta function of G at p is

$$Z^{\mathrm{oc}}_{G,p}(T) = \sum_{n=0}^{\infty} o_{p,n}(G)T^n.$$

Here $o_{p,n}(G)$ counts a certain family of orbits associated with G at level p^n. The exact meaning of the orbits depends on the counting problem: they may come from conjugacy classes, representation growth, or another natural equivalence relation.

The main point is that the sequence

$$o_{p,0}(G), o_{p,1}(G), o_{p,2}(G), \ldots$$

is packaged into a generating function.

This is similar in spirit to subgroup zeta functions and representation zeta functions: instead of studying the counting sequence directly, we study the analytic or algebraic properties of its generating function.

5.3. **The fivefold path to zeta functions.** The talk described a fivefold path from algebraic objects to zeta functions.

(1) **Replace the group by a Lie algebra.**

Start with a group, often a unipotent group or a nilpotent group, and pass to its Lie algebra. Then tweak the counting problem accordingly.

This step is useful because Lie algebras are linear objects. Their brackets are bilinear maps, hence tensors.

(2) **Derive one or more tensors from the Lie algebra.**
The Lie bracket
$$[-,-]:\mathfrak{g}\times\mathfrak{g}\to\mathfrak{g}$$
is a tensor.

Other tensors may also appear, depending on the counting problem. For example, one may consider commutator maps, adjoint maps, or tensors encoding the action of one part of the Lie algebra on another.

(3) **Construct an algebro-geometric zeta avatar from the tensor.**
Based on the specific counting problem, one constructs a geometric object derived from the tensor.

This object may be:

- a variety;
- a scheme;
- a family of matrices;
- a rank locus;
- a p-adic integral;
- or another geometric avatar encoding the enumeration problem.

The zeta function is then built from this avatar.

(4) **Throw the kitchen sink at it.**
Once the problem is expressed geometrically, one can use many tools:

- algebraic geometry;
- p-adic integration;
- model theory;
- combinatorics;
- invariant theory;
- resolution of singularities;
- computational algebra.

The point is to use every available method to understand the resulting zeta function.

(5) **Infer something about the resulting zeta function.**
Finally, one tries to prove structural properties of the zeta function, such as:

- rationality;
- functional equations;
- uniformity in p;
- pole structure;
- topological or reduced limits;
- dependence on tensor invariants.

5.4. **Why tensors appear.** Tensors appear because many algebraic structures are defined by multilinear operations.

For example:
$$[-,-]:\mathfrak{g}\times\mathfrak{g}\to\mathfrak{g}$$
is a bilinear operation, and therefore can be represented by a tensor.

Similarly, the commutator map of a class-2 nilpotent group gives an alternating bilinear map:
$$G/Z(G)\times G/Z(G)\to[G,G].$$

Thus, many group-theoretic counting problems can be recast as counting problems attached to tensors.

This is useful because tensor isomorphism and weaker equivalence relations between tensors can imply identities between the associated generating functions.

5.5. **Zeta functions as generating functions.** A zeta function in this context is a generating function encoding an algebraic counting problem.

Examples include:

- subgroup zeta functions;
- subalgebra zeta functions;
- ideal zeta functions;
- representation zeta functions;
- conjugacy class zeta functions;
- orbit-counting zeta functions.

Rossmann's `Zeta` package for SageMath implements methods for computing local and topological zeta functions arising from enumeration of subalgebras, ideals, submodules, representations, conjugacy classes, and related structures. :contentReference[oaicite:2]index=2

5.6. **Standard algebraic geometry.** Standard algebraic geometry provides tools to construct and study these zeta avatars.

For example, rank conditions on matrices define determinantal varieties. More generally, tensor-derived conditions often define algebraic varieties or schemes.

Once the counting problem has been transformed into geometry, one can ask:

- What is the dimension of the relevant variety?
- Is it irreducible?
- What are its singularities?
- How does it behave over finite fields?
- How do point counts over finite rings vary with p?

These geometric questions then influence the behavior of the zeta function.

5.7. **Topological zeta functions.** Another important idea in Rossmann's work is the topological version of such zeta functions.

Roughly speaking, topological zeta functions extract common features of local p-adic zeta functions as p varies, or in a limiting sense as $p \to 1$.

Rossmann introduced topological representation zeta functions for unipotent algebraic groups over number fields, inspired by Igusa's local zeta functions. These invariants capture common features of local representation zeta functions associated with pro-p groups derived from unipotent groups. :contentReference[oaicite:3]index=3

This fits the general theme of the talk: instead of studying each local counting problem separately, one tries to identify the geometric or tensorial structure responsible for the whole family of zeta functions.

5.8. **Takeaway.** The main takeaway is that tensor language provides a bridge between algebraic enumeration and zeta functions. The general pipeline is:

$$\text{group or algebra} \longrightarrow \text{Lie algebra} \longrightarrow \text{tensor} \longrightarrow \text{geometric avatar} \longrightarrow \text{zeta function}.$$

For the broader workshop, this means that tensor isomorphism is not only a classification problem. It can also control identities between generating functions.

Notes by Maksym Zubkov