Test Description

This document describes a test script located at:

/home/tommy/Documents/Maths/2025-Stage-M1/quat_alg_project/tests_max_order/test_max_order

We consider the following setup:

Let A be a finite-dimensional algebra over $\mathbb{Q}$, of dimension 16, defined by a $\mathbb{Q}$-basis $a_1, \ldots, a_{16}$. The multiplication is described via structure constants $c_{ijk} \in \mathbb{Q}$, satisfying:

$$a_i a_j = \sum_{k=1}^{16} c_{ijk} a_k$$

Our goal is to compute a *maximal order* in A, starting from a $\mathbb{Z}$-order given by the lattice:

$$\mathcal{O} = \mathbb{Z}a_1 + \cdots + \mathbb{Z}a_{16}$$

Motivation and Observations

The performance and difficulty of the maximal order computation heavily depend on the structure constants c_{ijk}. We distinguish two cases:

1. Bad Case (Hard)

If the structure constants are "nasty", i.e., contain large rational numbers and very few zero entries, then computing the left order of the lattice $\mathcal{O}$ becomes computationally expensive.

In practice, this can be diagnosed using:

```
sage:  A.table()
```

This returns a list of 16 matrices of size 16×16, one for each multiplication by a_i. When these matrices have large coefficients and are dense (not sparse), the algorithm suffers from:

- Large rational coefficients in intermediate steps.
- A large discriminant for the resulting order.
- Possible loss of precision or slow computations.

This situation typically occurs when the given basis $a_1, \ldots, a_{16}$ is the "natural" basis and no simplification is available.

2. Easiest case

When the structure constants are sparse (many zeros), the algorithm performs significantly better. The best-case scenario is when:

$$A \cong M_4(\mathbb{Q})$$

and the basis consists of the standard matrix units:

$$a_1, \ldots, a_{16} = E_{11}, E_{12}, \ldots, E_{44}$$

with multiplication rules:

$$E_{ij}E_{kl} = \delta_{jk}E_{il}$$

In this case, `A.table()` returns very sparse $16{\times}16$ matrices with mostly zeros, and the computation of the maximal order is efficient.

3. Good case ($\mathbf{C} = \mathbf{A} \otimes B^{\text{op}}$)

For our purpose of finding isomorphisms of quaternion algebras $A = (a, b \mid \mathbb{Q})$ and $B = (c, d \mid \mathbb{Q})$ given with their natural bases, the structure constants of $C = A \otimes B^{\text{op}}$ in the tensor product are not too bad.

For example:

```
sage: A = QuaternionAlgebra(QQ,1,-2)
sage: structure_constants(A,A.basis())
[
[1 0 0 0]   [ 0  1  0  0]   [ 0  0  1  0]   [0 0 0 1]
[0 1 0 0]   [ 1  0  0  0]   [ 0  0  0  1]   [0 0 1 0]
[0 0 1 0]   [ 0  0  0 -1]   [-2  0  0  0]   [0 2 0 0]
[0 0 0 1],  [ 0  0 -1  0],  [ 0 -2  0  0],  [2 0 0 0]
]

sage: B = QuaternionAlgebra(QQ,-2,-3)
sage: structure_constants(B,B.basis())
[
[1 0 0 0]   [ 0  1  0  0]   [ 0  0  1  0]   [ 0  0  0  1]
[0 1 0 0]   [-2  0  0  0]   [ 0  0  0  1]   [ 0  0 -2  0]
[0 0 1 0]   [ 0  0  0 -1]   [-3  0  0  0]   [ 0  3  0  0]
[0 0 0 1],  [ 0  0  2  0],  [ 0 -3  0  0],  [-6  0  0  0]
]

sage: Bop = opposite(B)
sage: C = tensor(A,Bop)
sage: C.table()
...
```

The output above demonstrates the structure constants of the tensor algebra $C = A \otimes B^{\text{op}}$, where the multiplication rules are manageable for computation. This setup simplifies the study of isomorphisms between quaternion algebras by allowing us to work in a larger, explicitly computable algebra.

Conclusion

This test highlights the importance of choosing a good basis for the algebra A when computing orders from structure constants. Sparse representations can drastically reduce the computational burden, whereas dense and complex constants can make the process practically infeasible.