AN ENHANCEMENT TO CHOP

By

ARI JOSEPH KUTOB BORMANIS

_____

A Thesis Submitted to The Honors College

In Partial Fulfillment of the Bachelors degree
With Honors in

Mathematics

THE UNIVERSITY OF ARIZONA

M A Y 2 0 2 1

Approved by:

_____

Dr. Klaus Lux
Department of Mathematics

Abstract

CHOP is a package by Max Neunhöffer and Felix Noeske in computational group theory implemented in the computer algebra system GAP. The main algorithm in CHOP, which we also will call CHOP, finds all of the irreducible modules of a given module of an algebra and is the central focus of this paper. First, the theory behind the algorithm and why it works is explained. Then a potential improvement to the algorithm for special cases is suggested. This improvement is explored numerically and the degree to which it was successful is discussed.

1 Introduction

Finding irreducible modules of a given algebra is an interesting question to mathematicians and an important one for physicists. Algebras arise naturally in physics as a result of trying to understand a system in terms of the symmetries inherent to that system. A prime example is studying the properties of a molecule by looking at its symmetries. Often such problems result in having to work with a matrix that can be quite large. Knowing the irreducible modules for the algebra underlying the system allows one to often simplify this matrix by means of a base change. The matrix created from this process can then be used to do computations that may have been difficult or infeasible with the old matrix. Specifically, the problem of finding eigenvalues is simplified in comparison to the original matrix.

One particular problem that arises from trying to study algebras computationally is that there often is not an easy way to represent them on a computer. This problem is of course avoided with matrix algebras as computers are able to efficiently deal with matrices. As such, it would be extremely useful to be able to take any algebra and work with it represented as a matrix algebra that was somehow equivalent. Representation theory gives tools for doing exactly that! Given an algebra A, a matrix representation of A is an algebra homomorphism from A to $M_n(R)$ (the matrix algebra of $n \times n$ matrices with entries from a ring R). For this it suffices to assign each generator of A a matrix in $M_n(R)$ such that this assignment yields a homomorphism. Of course, if the algebra A is not finitely generated this approach will not allow one to work with A computationally.

The computational algebra system GAP has a large body of code that has been developed and optimized for row vector-matrix calculations. This is the coding system that the algorithm package CHOP has been implemented in and will be used throughout the rest of the paper. The CHOP package can be used to determine a wide variety of information on an algebra, but this paper will be specifically focusing on the algorithm used to determine irreducible modules of an algebra. How the algorithm works is essentially as follows:

1. Suppose we are given a module $V = \mathbb{F}^{1\times n}$ by matrices in an algebra $A \subseteq \mathbb{F}^{n\times n}$, with $\mathbb{F}$ a finite field. Find an element $B \in A$ with small nontrivial kernel.

2. Apply a criterion to B to determine if the natural module $\mathbb{F}^{1\times n}$ is irreducible or there is a sub module.

3. If the module is irreducible stop, else a submodule has been found.

4. Call the submodule W, compute the action on W and V/W and recurse on both.

The enhancements provided in this paper mainly focus on steps 1 and 2. In the current CHOP implementation the algorithm seeks to find an element B with a *small* nontrivial kernel. However, it would seem in certain cases there may be a faster method. Namely, if the module has large submodules it may instead be faster to find these sub modules and recurse on them. It would seem like this second method can be achieved by computing the order polynomial of B and a random vector v from our module and then examining and doing computations with the factors of this polynomial. The goal of this paper is to describe the implementation of these improvements and seek to quantify to what extent they were successful.

2 Background Theory: Abstract Algebra

2.1 Basic Definitions

First, to talk about algebras first we must have the definition of an algebra. Specifically, the algebras we are interested in are K-algebras, where K is some field

Definition (K-algebra)**.** Let A be a vector space over K equipped with the additional binary operation $\cdot : A \times A \to A$ and let there be $1 \in A$ such that $1 \cdot v = v \cdot 1 = v$ for all $v \in A$. Then A is a K-algebra if the following hold for all x, y, z ∈ A and all a, b ∈ K:

1. Right distributivity: $(x+y) \cdot z = x \cdot z + y \cdot z$
2. Left distributivity: $z \cdot (x+y) = z \cdot x + z \cdot y$
3. Associativity: $(x \cdot y) \cdot z = x \cdot (y \cdot z)$
4. Compatibility with scalars: $(ax) \cdot (by) = (ab)(x \cdot y)$

Heuristically, the advantage of using K-algebras is that they allow one to use techniques from abstract algebra, because they share many similarities with rings, and they allow one to use techniques from linear algebra, since the underlying object is a vector space. Now for some important examples of algebras.

Example (Matrix algebra)**.** Given a non-negative integer $n \neq 0$ and a field R the standard matrix algebra is

$$M_n(R) = \{n \times n \text{ matrices with entries from } R\}$$

With the usual matrix operations of addition, multiplication, and scalar multiplication from elements of R.

Example (Endomorphism Algebra)**.** Let M be a vector space over K, we define the endomorphism algebra of M over K to be

$$\mathrm{End}_K(M) = \{\text{functions } f : M \to M | f \text{ is a linear map}\}$$

This forms an algebra under the operations of point-wise function addition, function composition for multiplication, and point-wise scalar multiplication for scalar multiplication.

Example (A commutative algebra)**.** Note that $\mathbb{C}$ can be viewed as a two dimensional vector space over $\mathbb{R}$ with basis $\{1, i\}$, but the multiplication of $\mathbb{C}$ also makes it a commutative $\mathbb{R}$-algebra.

Example (A finite algebra)**.** Let $\mathbb{F}_q$ denote the finite field of cardinality q, where $q = p^n$ for p a prime and n a positive integer. Consider $\mathbb{F}_{7^3}$ over $\mathbb{F}_7$. We have that $\mathbb{F}_{7^3}$ is a three dimensional vector space over $\mathbb{F}_7$, but ring multiplication also makes it into a commutative $\mathbb{F}_7$-algebra.

Next, it is natural to consider what constitutes a sub-object of an algebra.

Definition (Subalgebra)**.** A subset L of a K-algebra A is a subalgebra if L is a K-linear subspace of A that is closed under multiplication.

Example. We see that $\mathbb{C}$ is an $\mathbb{R}$-algebra. A subalgebra of $\mathbb{C}$ is $\mathbb{R}$ itself, as clearly $\mathbb{R}$ is closed under multiplication and addition.

Using this definition we may now define what it means for a subalgebra to be generated by some set, but first we need the following important result.

Theorem (Intersection of subalgebras)**.** Let A be a K-algebra and $S = \{L_i | i \in I\}$ be a collection of subalgebras of A. Then $L = \bigcap_{i \in I} L_i$ is a subalgebra of A.

Proof: First note that L is nonempty as it must contain 0 since all subalgebras are also linear subspaces. Let $x, y \in L$, $c \in K$, and $i \in I$. As $x, y \in L_i$ and L_i is a subalgebra we must have $x+y \in L_i$, $xy \in L_i$, and $xc \in L_i$. Hence $x+y \in \bigcap_{i \in I} L_i$, $xy \in \bigcap_{i \in I} L_i$, and $cx \in \bigcap_{i \in I} L_i$. Therefore L is a subalgebra. □

We may now define what it means for a subalgebra to be generated by a set.

Definition (Subalgebra generated by a set)**.** Let S be a subset of a K-algebra A. The algebra generated by S is the intersection of all subalgebras in A containing S.

Example (Subalgebra generated by a single element)**.** Let A be a K-algebra with $a \in A$. Consider L_a, the subalgebra generated by $\{a\}$. Certainly $a \in L_a$ by definition, and since L_a is closed under multiplication we must have that all powers of a are in L_a. Furthermore, since L_a is closed under scalar multiplication we must have that $ca^i \in L_a$ for all $c \in K$ and $i \in \mathbb{Z}_{>0}$. Thus we have

$$K[a] = \{c_1 a^{i_1} + c_2 a^{i_2} + ... + c_n a^{i_n} | i_1, i_2, ..., i_n \in \mathbb{Z}_{>0}, n \in \mathbb{Z}_{>0}, c_1, c_2, ..., c_n \in K\} \subseteq L_a$$

Now the question is if $K[a] = L_a$? First note that K[a] is in fact a subalgebra, which follows directly from definition. If we were to have $b \in L_a$ such that b was not of the form $c_1 a^{i_1} + c_2 a^{i_2} + ... + c_n a^{i_n}$ then it would follow $b \notin K[a]$. However, K[a] is a subalgebra containing a. Since L_a is defined to be the intersection of all subalgebras containing a we would then have $b \notin L_a$! It follows that $K[a] = L_a$.

Please note that this paper is mainly concerned with finitely generated algebras (i.e. S is finite) as these are the most practical to work with computationally. Next, we must define a homomorphism of algebras as this is the foundation of representation theory for algebras.

Definition (K-algebra homomorphism)**.** Given two K-algebra's A and B, a function $f : A \to B$ is a K-algebra homomorphism if f is a K-linear map and for all $x, y \in A$, $f(xy) = f(x)f(y)$

Example. If M is a K-vector space with $\dim(M) = n$ then there is a K-algebra homomorphism $f : \mathrm{End}_K(M) \to M_n(K)$ where f sends linear maps to their matrix representation in $M_n(K)$.

Now we may define a representation of an algebra.

Definition (Representation of an algebra)**.** Given a K-algebra A, a representation of A is a pair (M, f), where M is a K-vector space and $f : A \to \mathrm{End}_K(M)$ is an algebra homomorphism.

The M mentioned in the definition above is actually an example of a module, which we will now define.

Definition (Left A-module)**.** Let A be an algebra. A left A-module M consists of an abelian group $(M, +)$ with operation $\cdot : A \times M \to M$ such that for all $a, b \in A$ and $m, n \in M$:

1. $a \cdot (m + n) = a \cdot m + a \cdot n$
2. $(a + b) \cdot m = a \cdot m + b \cdot m$
3. $(ab) \cdot m = a \cdot (b \cdot m)$
4. $1_A \cdot m = m$

In this way we see that giving a module is the same as giving a representation, since we may define an algebra homomorphism $f : A \to \mathrm{End}_K(M)$ by sending $a \in A$ to the function $f(a)(m) = a \cdot m$ for all $m \in M$. Likewise giving a representation defines a module. If our representation is given by $f : A \to \mathrm{End}_K(M)$ we may define $\cdot : A \times M \to M$ by $a \cdot m = f(a)(m)$. Note that module homomorphisms are just linear maps. Finally, we give the definition of submodule, quotient module, and an irreducible module. The latter being the mathematical object we are interested in finding.

Definition (Submodule)**.** Let M be a left A-module and N and subgrop of M. Then N is a submodule of M if for all $n \in N$ and all $a \in A$, we have $a \cdot n \in N$.

Definition (Quotient Module)**.** Let M be a left A-module and N a submodule of M. Then the quotient module M/N is the quotient group M/N equipped with the scalar multiplication for all $a \in A$ and $n \in N$, $a(n + N) = an + N$. This scalar multiplication makes M/N into a left A-module.

Definition (Irreducible Module)**.** Let M be a left A-module. Then M is said to be irreducible if the only submodules of M are itself and $\{0\}$.

2.2 Matrix Representations

We are interested in a specific kind of representation known as a matrix representation. Essentially, this allows us to work with our algebra as a collection of matrices, which makes it easier do computations with the algebra.

Definition (Matrix representation)**.** Given a K-algebra A, a matrix representation of A is a pair ($M_n(K)$, f) where f is an algebra homomorphism $f : A \to M_n(K)$.

One would always like to be able to work with algebras as matrices, so one would hope a matrix representation would always exist. Indeed, such a representation does always exist! We can prove this using our notion of a standard representation. Let A be an algebra and M a left-module of A. Define $f : A \to \mathrm{End}_K(M)$ where f(a) is the linear map $a \cdot m$ for all $a \in A$ and $m \in M$. First note that certainly $f(a) \in \mathrm{End}_K(M)$ since for all $m, n \in M$ and $k \in K$

$$f(a)(m+n) = a \cdot (m+n) = a \cdot m + a \cdot n = f(a)(m) + f(a)(n)$$

Which is just using the definition of a module. Next,

$$f(a)(km) = a \cdot (km) = (ak)m = k(am) = kf(a)(m)$$

Which uses compatibility with scalars. So indeed $f(a) \in \mathrm{End}_K(M)$ for all $a \in A$. Now we must verify that f is an algebra homomorphism. Let $a, b \in A$, $k \in K$, and $m \in M$

$$f(ka)(m) = (ka) \cdot m = k \cdot (a \cdot m) = kf(a)(m)$$

As this hold for all $m \in M$ we have that f(ka) = kf(a). Next we see

$$f(ab)(m) = (ab) \cdot m = a \cdot (b \cdot m) = f(a)f(b)(m)$$

Here remembering that multiplication in $\mathrm{End}_K(M)$ is function composition. So indeed f(ab) = f(a)f(b). Lastly,

$$f(a+b)(m) = (a+b) \cdot m = a \cdot m + b \cdot m = f(a)(m) + f(b)(m)$$

Therefore, f(a + b) = f(a) + f(b) and so f is an algebra homomorphism and (M, f) is a representation of A. Now we consider $g : \mathrm{End}_K(M) \to M_n(K)$ with n = dim(M) and $g(\rho) = M_\rho$ where M_ρ is the matrix representation of $\rho \in \mathrm{End}_K(M)$. The properties of matrix representations and high level linear algebra then yield that g is an algebra isomorphism. It is then easily verified that the composition of algebra homomorphisms is another algebra homomorphism. Hence, $g \circ f : A \to M_n(K)$ is an algebra homomorphism and a matrix representation of A.

3 Background Theory: CHOP

A key result of the last section was that all algebras have a matrix representation. With this established, we change our focus from finding matrix representations to what we can do with a matrix representation once we have one. For the purposes of CHOP, we are given a matrix representation without any knowledge of what algebra it is representing. The goal now is to find the irreducible modules of this matrix representation. First, we explain several key processes CHOP uses and then give a proof of why CHOP works.

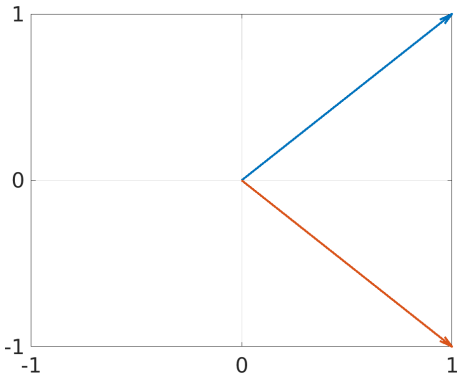
3.1 Spinning

Perhaps the most fundamental process to CHOP is that of "spinning" a vector. This is a process by which one vector (that is not equal to zero), v, is used to make a basis for the smallest subspace containing v that is invariant under the matrices with which v is "spun". Before going further, consider the following example in $\mathbb{R}^2$.

Example. Consider the following vector in $\mathbb{R}^2$, $[1,1]^t$. Say that we want to find the smallest subspace containing v that is invariant under the matrix $\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$. One algorithmic way to do this is to multiply $[1,1]^t$ by said matrix and in essence "spin" it. Indeed we see that

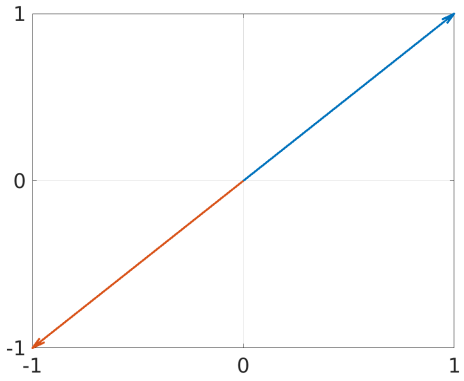
$$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Visually,



So we see that we have rotated our vector 90 degrees and in doing so created the basis $\{[1,1]^t, [1,-1]^t\}$, which in this case is a basis for the whole space, but what if our choice of matrix was different? For example

$$\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \end{bmatrix}$$



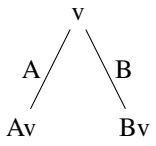
Now we see that we spun or original vector 180 degrees. No matter how much we continue to spin the vector (using our current matrix) we will never leave the subspace. It follows that the smallest subspace containing v that is invariant under $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ is span$\{v\}$.

This example gives a good geometric intuition for why this process is called "spinning" a vector and also highlights two important outcomes of spinning: one, the process can create a basis of a proper subspace, two, the process can create a basis of the entire space. A small warning, the geometric way of thinking about spinning is mainly useful for having a mental picture of the process. Indeed, there are many spaces in which the geometric concept of "spinning" does not make much sense. It is also clear to see that if the vector is an eigenvector of the matrix we are spinning it under that it gets "stretched" or "compressed" instead of spun.

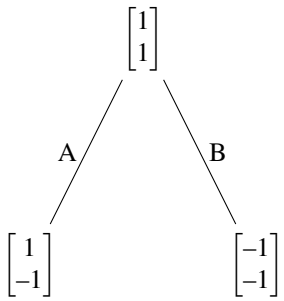
Instead, a good way to work with and make sense of spinning vectors is through trees (in the computer science sense). Indeed, following our previous example we may define

$$A = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, B = \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}, v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

and then we may spin under both A *and* B. To keep track of this process we use a tree like so



Or equivalently,



Algorithmically, we proceed as following: first, we start with a single nonzero vector v and define $\mathbb{B} = \{v\}$. Then we multiply v by all matrices we are spinning under, which in this case is just A and B. Doing this forms the vectors Av and Bv. Next we check to see whether Av and Bv are linearly independent from all vectors in $\mathbb{B}$. Clearly, Bv is a scalar multiple of v so it is not linearly independent from all vectors in $\mathbb{B}$. On our tree, that means we just stop at the node containing Bv. Now Av is linearly independent from v, so we add it $\mathbb{B}$ to form $\mathbb{B} = \{v, Av\}$. We could continue from the node containing Av to form the nodes AAv and BAv, but we see this is unnecessary as clearly $\mathbb{B}$ now constitutes a basis for $\mathbb{R}^2$ and so any further spinning will just produce linearly dependent vectors.

In CHOP we spin under the generators of our matrix algebra. Therefore, the space produced will be invariant under these generators. Lastly, note that we use the notation $B = (a_1, ..., a_n)$ to denote an ordered list, where the entries of the list are indexed starting from 1. The notation B(i) is used to denote the ith entry of B, and in our current example, is only defined for $1 \leqslant i \leqslant n$. Now we may state the full spinning algorithm as follows: given a matrix algebra $A = \langle A_1, A_2, ..., A_n \rangle \subseteq \mathbb{F}^{n\times n}$ (where this notation means A is generated by $\{A_1, A_2, ..., A_n\}$) and $v \in \mathbb{F}^{n\times 1}$ such that $v \neq 0$ we may spin v under A by doing the following

1. Define $\mathbb{B} = (v)$ and set $i = 1$.
2. Compute A_1v, A_2v, ..., A_nv.
3. For $1 \leqslant j \leqslant n$ check if $A_jv \in \text{span}(\mathbb{B})$. If $A_jv \notin \text{span}(\mathbb{B})$ append A_jv to $\mathbb{B}$.
4. Set $i = i + 1$. If the length of $\mathbb{B}$ is less than i stop. Else, set $v = B(i)$ and repeat steps 2 through 4.

Note that to guarantee this process terminates it is necessary that the space containing v is finite dimensional. Indeed, given that $\mathbb{F}^{n\times 1}$ has dimension $n < \infty$, it is easy to see that this algorithm terminates after a finite number of steps. The process will stop when all leaves of the tree are linearly dependent with respect to $\mathbb{B}$. The max length of a linearly independent set in $\mathbb{F}^{n\times 1}$ is $\dim(\mathbb{F}^{n\times 1}) = n$ and so it follows the algorithm must stop after a finite number of steps.

3.2 Extending a Basis

One of the ubiquitous processes in CHOP (along with spinning vectors) is that of extending a linearly independent set of vectors to a basis for the whole space. Said differently, once a submodule is found we wish to extend its basis to a basis for the entire module. This is so that we may compute the action of the space on the quotient module formed by quotienting the space by the submodule. Furthermore, we want the process of basis extension to be algorithmic so that we can code it in a computer.

For this purpose we may consider a "cleaned" set of vectors. That is, all of the non-pivot entries have been zeroed out by the pivot entries. This is analogous to a row reduced matrix. The fact that we may consider this slightly nicer basis follows directly from the linear algebra theoretic result that this basis spans the same space as the original. With this in mind, we begin the proof.

Theorem (Basis extension)**.** Let $B=\{b_1, b_2, ..., b_m\}$ be a linearly independent set of vectors of $\mathbb{F}^{n\times 1}$, with $m < n$. Then we may extend B to a basis of $\mathbb{F}^{n\times 1}$. Furthermore, we may do this by finding all of the pivots of B and then adding a vector to B for each non-pivot. Where the vector we add has zeros in all entries except for the non-pivot.

Proof: Let $B=\{b_1, b_2, ..., b_m\}$ be a linearly independent set of vectors of $\mathbb{F}^{n\times 1}$, with $m < n$. Assume without loss of generality that for $b_i \in B$, b_i has a nonzero entry in the ith position, call this entry c_i, and all other vectors in B have a zero in the ith position. Define the canonical basis vectors to be the vectors in $\mathbb{F}^{n\times 1}$ with 1 in exactly one position and zeros in all other entries. Let e_j for $1 \leqslant j \leqslant n$ denote the canonical basis vector with 1 in the jth position and zeros everywhere else. Extend B with the canonical basis vectors to $B^* = \{b_1, b_2, ..., b_m, e_{m+1}, ..., e_n\}$. I claim B^* is a basis for V. Let $v = [v_1, v_2, ..., v_n]^t \in V$, we will try to find a_1, a_2,..., $a_n \in \mathbb{F}$ such that

$$a_1 b_1 + a_2 b_2 + ... + a_n e_n = v$$

$$a_1 \begin{bmatrix} c_1 \\ 0 \\ \vdots \\ * \end{bmatrix} + a_2 \begin{bmatrix} 0 \\ c_2 \\ \vdots \\ * \end{bmatrix} + ... + a_n \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

For a_k with $1 \leqslant k \leqslant m$ it is clear that $a_k = v_k c_k^{-1}$. Next, let $d_{i,j}$ be the value of b_i in the jth position if $1 \leqslant i \leqslant m$, and if $m+1 \leqslant i \leqslant n$, let $d_{i,j}$ be the value of e_i in the jth position. For each a_k with $m+1 \leqslant k \leqslant n$ we have an equation of the form

$$\sum_{i=1}^{k-1} a_i d_{i,k} + a_k = v_k$$

This implies

$$a_k = v_k - \sum_{i=1}^{k-1} a_i d_{i,k}$$

We then have that B^* spans V and is a set of exactly n vectors. Therefore, by theorem, B^* is a basis of V. $\square$

Example. Consider the space $\mathbb{F}_3^4$ and say we want to extend the set $B = \{[1,0,2,1]^t, [1,1,1,1]^t\}$ to a basis of the whole space. By the theorem above all we need to do is find the pivots of B. For this purpose we may put the two vectors in B together in a matrix and *column* reduce (i.e we may only add and subtract the columns of our matrix).

$$\begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 2 & 1 \\ 1 & 1 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 2 & 2 \\ 1 & 0 \end{bmatrix}$$

Where for the 2 times column 1 was added to column 3. Thus we see that we are missing pivots in the third and fourth rows. By the theorem above we may add $[0,0,1,0]^t$ and $[0,0,0,1]^t$ to our basis to complete it to a basis of the whole space.

3.3 Norton's Criterion

The following is a result of the brilliant mathematician Simon P. Norton. This is the keystone of why CHOP works. Note that here we make the change to working with row vectors and the row space $V = \mathbb{F}^{1\times n}$. GAP is actually optimized to work with row vectors, but from a mathematical standpoint working with row vectors is equivalent to working with column vectors. The examples before were given with column vectors since it is assumed that a majority of readers are more comfortable working with them.

Theorem (Norton's Irreducibility Criterion)**.** Let $A = \langle A_1, A_2, ..., A_k \rangle \subseteq \mathbb{F}^{n\times n}$ be a matrix algebra and $B \in A$ a singular element ($\det(B) = 0$). Let $A^t = \langle A_1^t, A_2^t, ..., A_k^t \rangle$ and $V = \mathbb{F}^{1\times n}$. At least one of the following holds:

1. There is a $0 \neq v \in \ker(B)$ such that $vA = \{vX \mid X \in A\} \neq V$.
2. For all $v \in \ker(B^t)$ it holds that $vA^t \neq V$
3. The natural module $V = \mathbb{F}^{1\times n}$ is irreducible.

Proof: Assume that 1 and 3 do not hold. So there is some invariant subspace W of V with $W \neq \{0\}$. Let $\dim(W) = e$. We can choose some basis $\{w_1, ..., w_e\}$ of W and extend it to a basis $\{w_1, ..., w_e, v_1, ..., v_{n-e}\}$ of V. We may write all matrices with respect to this basis and let T be the matrix $T = [w_1|...|w_e|v_1|...|v_{n-e}]$. This denotes that if $1 \leqslant i \leqslant e$ then the ith row of T is w_i and if $e < i \leqslant n-e$ then the ith row of T is v_{i-e}. Define $B' = TBT^{-1}$ and note that this means B' is also singular. Since W is invariant, B' looks like this:

$$B' = \begin{bmatrix} M & 0 \\ * & N \end{bmatrix}$$

Where $M \in \mathbb{F}^{e\times e}$ and $N \in \mathbb{F}^{(n-e)\times(n-e)}$. Since 1 does not hold we must have $\ker(B') \bigcap W = 0$. Else there would be some $v \in \ker(B') \bigcap W$ with $v \neq 0$, and since 1

does not hold it must be that $vA = V$, contradicting that W is invariant. So $\ker(B') \bigcap W = 0$ and we claim this implies M has full rank e. Let $w \in W$. Since W is invariant w must have the form $w = \begin{bmatrix} w_0 & \mathbf{0} \end{bmatrix}$ where $w_0 \in \mathbb{F}^{1\times e}$ and $\mathbf{0} \in \mathbb{F}^{1\times(n-e)}$. Hence,

$$w \cdot B' = \begin{bmatrix} w_0 & \mathbf{0} \end{bmatrix} \cdot \begin{bmatrix} M & 0 \\ * & N \end{bmatrix} = \begin{bmatrix} w_0 \cdot M + \mathbf{0} \cdot * & w_0 \cdot 0 + \mathbf{0} \cdot N \end{bmatrix} = \begin{bmatrix} w_0 \cdot M & \mathbf{0} \end{bmatrix}$$

If $w_0 \neq 0$ it follows $w_0 \cdot M \neq 0$. Else, we would have that $w \in W$ and $w \in \ker(B')$ with $w \neq 0$, a contradiction. So for all $u \in \mathbb{F}^{1\times e}$, $u \cdot M = 0$ only if $u = 0$, and hence M has full rank. Next let $v \in \ker(B'^t)$ and write $v = \begin{bmatrix} v_0 & v_1 \end{bmatrix}$, where $v_0 \in \mathbb{F}^{1\times e}$ and $v_1 \in \mathbb{F}^{1\times(n-e)}$. Then

$$0 = v \cdot B'^t = \begin{bmatrix} v_0 & v_1 \end{bmatrix} \cdot \begin{bmatrix} M^t & * \\ 0 & N^t \end{bmatrix} = \begin{bmatrix} v_0 \cdot M^t & v_0 \cdot * + v_1 \cdot N^t \end{bmatrix}$$

It follows that $v_0 = 0$ and so all vectors in $\ker(B'^t)$ have the form $v = \begin{bmatrix} \mathbf{0} & v_1 \end{bmatrix}$, with $\mathbf{0} \in \mathbb{F}^{1\times e}$. Therefore, for all $v \in \ker(B'^t)$ it holds that $vA^t \neq V$. □

Here we begin to develop insight into how one can "hunt" irreducible modules. First note that there is an algorithm called the "word generator" in CHOP that will create the element B. Now if 1 and 2 fail then we know that we have found an irreducible module. Otherwise, either 1 or 2 must be true. If 1 is true then we have found some submodule of the form vA, which we can obtain by using the method of spinning vectors. Similarly, if 2 is true we have again found a submodule, but this time in the transpose of our algebra. Either way we may now try to find irreducible modules in the proper submodule and the quotient of the space by the proper submodule. In this way we may keep recursing until all irreducible submodules are found.

3.4 How Chop Functions

Now we are ready to describe a basic step of CHOP. Assuming we are given an A-module $V = \mathbb{F}^{1\times d}$ by matrices $A_1, ..., A_k \in \mathbb{F}^{d\times d}$ with $A = \langle A_1, ..., A_k \rangle$ we begin searching for irreducible modules of V as follows:

1. Find an element $B \in A$ with small nontrivial kernel. This is implemented in CHOP as the "word generator" function.

2. Compute ker(B) and start to spin all $0 \neq v \in \ker(B)$ under A to try and find a proper submodule of V. If a proper submodule vA of V is found compute matrices for the action of A on vA and V/vA and recurse.

3. Otherwise, spin one $0 \neq v \in \ker(B^t)$ under A^t.

4. If $vA^t = V$ then we have proved the module V is irreducible, stop. Else, compute matrices for the action of A^t on vA^t and V/vA^t, take the transpose of these matrices, and then recurse.

For some readers it might be vague what "the action of A on W and V/W" means. We essentially want to see how the generators of A move around the basis vectors of W and V/W. This information can be stored matrices and is shown in the next section.

3.5 An Example

To understand how Chop functions it is best to go through a concrete example. Specifically, we will be looking at a representation of a group algebra.

Definition (Group Algebra)**.** A group algebra of a group G over a field K is denoted K[G] and is comprised of finite linear combinations of elements in g with coefficients in K. Let $a \in K$ and

$$\sum_{g\in G} a_g g, \text{ and } \sum_{g\in G} b_g g$$

be two elements of K[G], where $a_g = 0$ and $b_g = 0$ for all but finitely many $g \in G$. We define the operations of the algebra as follows:

$$\sum_{g\in G} a_g g + \sum_{g\in G} b_g g = \sum_{g\in G} (a_g + b_g)g$$

$$\left[\sum_{g\in G} a_g g\right] \cdot \left[\sum_{g\in G} b_g g\right] = \sum_{g\in G,\, h\in G} (a_g b_h)gh$$

$$a \sum_{g\in G} a_g g = \sum_{g\in G} (aa_g)g$$

Now consider the group algebra given by $\mathbb{F}_2[S_3]$ (where S_3 is the symmetric group on three elements). Using the work in the previous sections we may represent $\mathbb{F}_2[S_3]$ with $\mathbb{F}_2^{3\times 3}$ by prescribing images to the generators of $\mathbb{F}_2[S_3]$. With regards to this note that $\{(12),(123)\}$ generates $\mathbb{F}_2[S_3]$. Indeed, $\{(12),(123)\}$ generates S_3 so clearly linear combinations of products in $(12),(123)$ can be used to generate $\mathbb{F}_2[S_3]$. For the sake of convenience we represent these generators as permutation matrices, but note that we could certainly represent these generators using other matrices in $\mathbb{F}_2[S_3]$.

$$(12) \mapsto \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = A, \qquad (123) \mapsto \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} = B$$

Next we must choose a basis for $\mathbb{F}_2^{3\times 3}$. We will choose $\mathbb{B} = \{[1,0,0]^t, [0,1,0]^t, [0,0,1]^t\} = \{e_1, e_2, e_3\}$ as this keeps computations simple. Now we try to find an element of $\mathbb{F}_2^{3\times 3}$ with small nontrivial kernel and then spin up the nonzero elements of the kernel to try and find a proper submodule. One such element is $A + B$, indeed

$$A + B = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

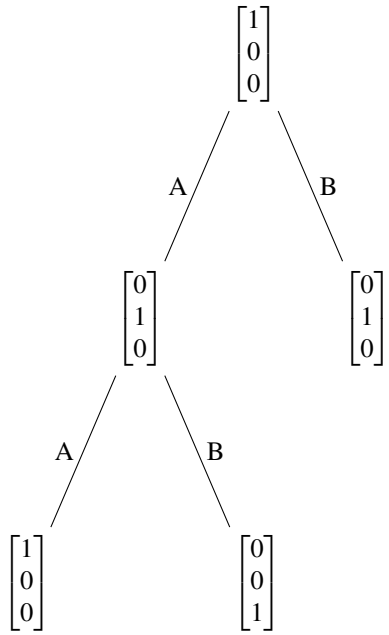
And changing the equation $(a+b)x = 0$ into an augmented matrix and solving yields

$$\left[\begin{array}{ccc|c} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{array}\right] \rightarrow \left[\begin{array}{ccc|c} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

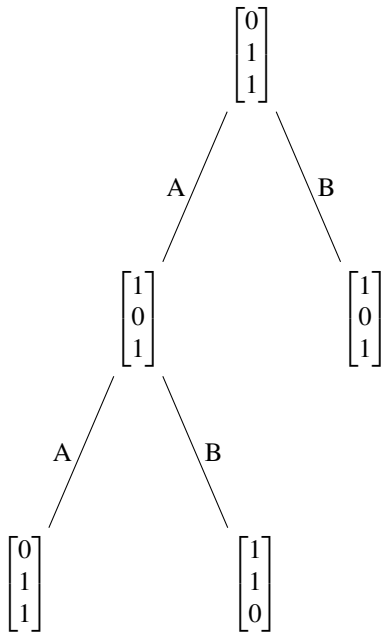
Letting $x = [x_1, x_2, x_3]^t$ this implies x_1 is free and $x_2 = -x_3 = x_3$ (using that $-1 = 1$ in $\mathbb{F}_2$). This allows use to concretely write out all elements of the kernel as follows

$$\ker(A+B) = \left\{ \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \right\}$$

It remains to spin up the nonzero elements of $\ker(a+b)$. Starting first with $[1,0,0]^t$ we see



As a convention, let the topmost level of the tree be level 1, the level directly beneath the topmost level be level 2, and so on. Here the labeling above each edge corresponds to whether the vector in the parent node was multiplied on the left by A or B. Looking at the first vector on level 1, either vector on level 2, and the leftmost vector on level 3 we see that we have the vectors e_1, e_2, and e_3. So the spinning process generated the whole space and we must select another vector in our kernel. This time spin up $[0,1,1]^t$ to get



Here we stop since all of the vectors on the third level are linear combinations of vectors from the preceding levels. Therefore we have discovered a submodule of $\mathbb{F}_2^{3\times 1}$ with basis given by $\mathbb{B}_1 = \{[0,1,1]^t, [1,0,1]^t\}$! To be able to recurse on the module and quotient module we first extend this basis to a basis for $\mathbb{F}_2^{3\times 1}$. Thinking of the two vectors in $\mathbb{B}_1$ as a matrix and looking for pivots makes it easy to see that adding $[0,0,1]^t$ to $\mathbb{B}_1$ extends it to a basis of the whole space. In pictures,

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix}$$

It follows that we need a pivot in the third row making $[0,0,1]^t$ a natural choice. We now have the basis $\mathbb{B}_1^* = \{[0,1,1]^t, [1,0,1]^t, [0,0,1]^t\}$ and compute the action of A and B on this basis. This will allow us to find the generators of the submodule. Let $u_1 = [0,1,1]^t$, $u_2 = [1,0,1]^t$, and $u_3 = [0,0,1]^t$. We compute the following:

$$Au_1 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = 0 \cdot u_1 + 1 \cdot u_2 + 0 \cdot u_3$$

$$Au_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} = 0 \cdot u_1 + 0 \cdot u_2 + 1 \cdot u_3$$

$$Au_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = 0 \cdot u_1 + 1 \cdot u_2 + 0 \cdot u_3$$

We then can compute the matrix representation of $\mathbb{B}_1^*$ with respect to this action easily as

$$M(A) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Repeating the same process for the action of B on $\mathbb{B}_1^*$ yields

$$M(B) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

So now we recurse with the matrices $A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ and $B_1 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$, which represent the action of the generators A, B on the proper submodule we found, and note that the action of A, B on quotient module is represented by $A_2 = B_2 = \begin{bmatrix} 1 \end{bmatrix}$. These matrices come from M(A) and M(B), A_1 is the upper left block on the diagonal of M(A), and likewise, B_1 is the upper left block on the diagonal of M(B). Similarly, A_2 and B_2 come from the lower right blocks on the diagonals of M(A) and M(B) respectively.

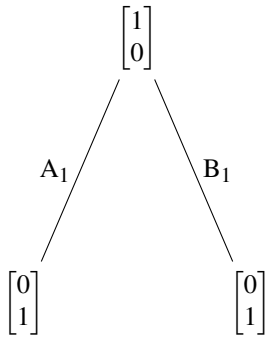
The quotient module is easy to determine, as $A_2 = B_2 = \begin{bmatrix} 1 \end{bmatrix}$ defines the trivial module. For the proper module, we repeat the same procedure as above. To find an element with an nonzero kernel we try

$$A_1 + B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

This clearly has a nonzero kernel, which we can easily compute.

$$\ker(A_1 + B_1) \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right\}$$

Now we spin up the only nonzero vector



The vectors on levels 1 and 2 clearly form a basis for $\mathbb{F}_2^{2\times 1}$ so we stop the spinning procedure. At this point we have exhausted all nonzero vectors in the kernel, so we

begin applying the second step of Norton's Criterion. For this second step we first transpose the matrices A_1, B_1, and $A_1 + B_1$.

$$A_1^t = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad B_1^t = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}, \qquad (A_1 + B_1)^t = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$$

We see that none of the matrices in question have actually changed. Therefore the kernel of $(A_1 + B_1)^t$ is the same as the kernel of $A_1 + B_1$ and we may again spin up $[1,0]^t$ to get the whole space. Since there is a vector in the kernel of $(A_1 + B_1)^t$ that spins up to the whole space, and because no vector in the kernel of $A_1 + B_1$ that spins up to the whole space, by Norton's Criterion the module is irreducible.

At this point Chop would return the matrices (A_1, B_1) as a pair for the submodule and (A_2, B_2) for the quotient module.

4 Implementation of Improvements

Here we describe the theoretical goal of the proposed implementation and why it should work. The main class of examples that the code was tested on is regular modules of alternating groups over finite fields. The reason for this is twofold: one, regular modules tend to be highly reducible making them a rich class of examples to test the code on. Two, much of the theory of regular modules is already known. This makes it easier to give theoretical predictions for the code. In this section pseudo-code and code for implementation is also given and the results of the code are numerically explored.

4.1 Minimal Polynomials and Order Polynomials

It has been previously mentioned that CHOP normally functions by looking for singular elements with small kernels, but how does one create these elements? One answer is by using what is known as the minimum polynomial of a matrix.

Definition (Minimum Polynomial of a Matrix)**.** Given $A \in \mathbb{F}^{n\times n}$, the minimum polynomial of A over $\mathbb{F}$ is the lowest degree monic polynomial $P(x) \in \mathbb{F}[x]$ such that $P(A) = 0$.

Note that such a polynomial always exists and is unique. Now say that we have a matrix algebra $A \subseteq \mathbb{F}^{n\times n}$ and a module for A, $V = \mathbb{F}^{1\times n}$. We can easily find some $A_0 \in A$ by taking A_0 to be some expression in the generators of A. Let P_0 be the minimum polynomial of A_0 and say that P_0 factors as

$$P_0(x) = cq_1(x)^{i_1}q_2(x)^{i_2}...q_k(x)^{i_k}$$

with $c \in \mathbb{F}$, each $q_j(x)$ with $1 \leqslant j \leqslant k$ irreducible and $i_j \in \mathbb{Z}_{>0}$. Next, assume without loss of generality that $i_1 = \min\{i_1, ..., i_k\}$. Then it can be shown that $q_1(A_0)^{i_1}$ will have a relatively small kernel compared to other choices of factors. Furthermore, $q_1(A_0)^{i_1}$ is singular since P_0 is minimal and so

$$cq_2(A_0)^{i_2}...q_k(A_0)^{i_k} \neq 0$$

Thus there is $v \in V$ such that $v[cq_2(A_0)^{i_2}...q_k(A_0)^{i_k}] \neq 0$ and this is clearly in the kernel of $q_1(A_0)^{i_1}$. This is how one can in practice find elements with relatively small kernels. However, the implementation presented in this paper is actually interested in finding large kernels and does not use minimal polynomials. In practice, what is known as the order polynomial provides a good approximation for the minimum polynomial and is defined as

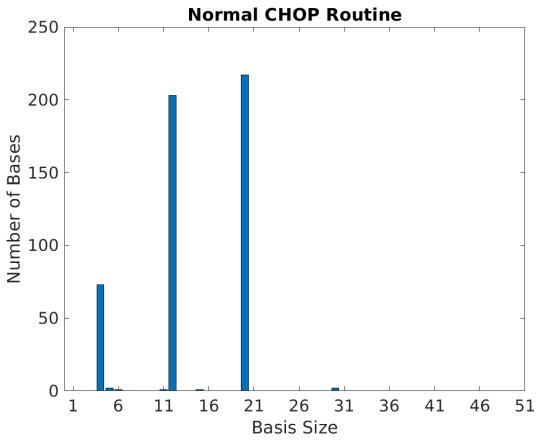
Definition (Order Polynomial of a Matrix)**.** Given $A \in \mathbb{F}^{n\times n}$ and $v \in \mathbb{F}^{1\times n}$, the order polynomial of A over $\mathbb{F}$ with respect to v is the lowest degree monic polynomial $P(x) \in \mathbb{F}[x]$ such that $vP(A) = 0$.

We may then go about the same procedure described above with A_0 but instead with some $0 \neq v \in V$ and computing the order polynomial of A_0 with respect to v. Call this order polynomial P_1, then we factor P_1 like before, but now we choose the irreducible factor with maximum multiplicity. We can then easily make a vector in the

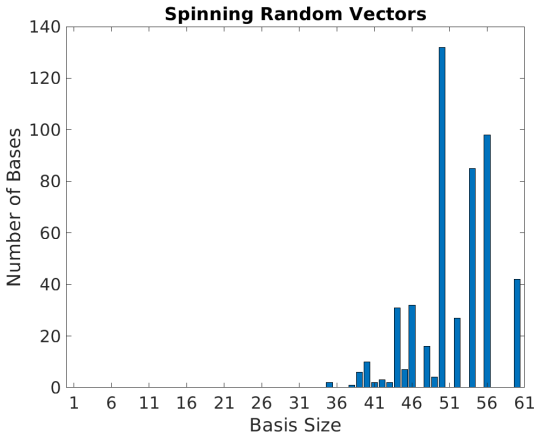
kernel of this factor raised to its multiplicity. Since the factor was chosen to have maximal multiplicity the hope is the vector we created would spin up to a relatively large subspace. Lastly, note that the reason we use the order polynomial over the minimal polynomial is that the minimum polynomial is computationally unwieldy; whereas, the order polynomial is far cheaper to compute.

4.2 A Numerical Exploration

Here we discuss the implementation of a potential improvement to Chop, how it should function, and a numerical exploration of how the implementation does function. First, it is pertinent to discuss how Chop normally functions. This is discussed in the previous section, but as a review, remember that Chop seeks elements of the matrix algebra with small non-trivial kernel and then begins spinning elements of the kernel. How this happens in practice is that an algebra element is selected, its minimal polynomial is computed, and factors with low multiplicity in the minimal polynomial are selected as candidates for spinning. Following this procedure tends to generate subspaces of relatively small dimension. Chop's normal element selection and spinning routine was isolated and ran 500 times to produce the following graph:



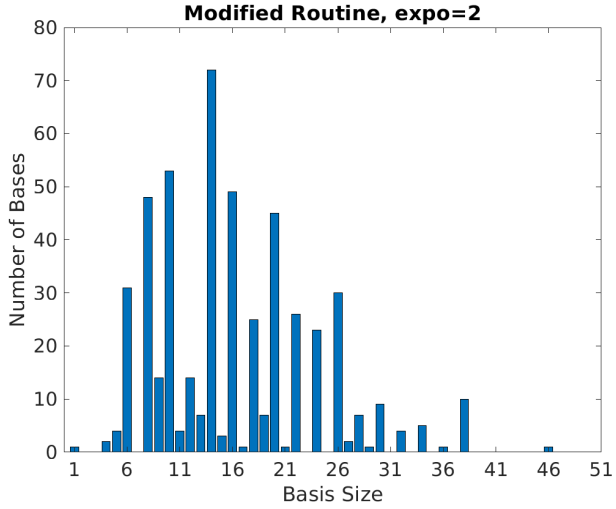
This was done using the regular module of the alternating group A_5 over the Galois Field with two elements, GF(2). This module has dimension 60, and as we can see from the graph above, Chop's normal subspace finding procedure yields subspaces that are approximately $\frac{1}{3}$ the size of the module. Compare this to trying to find subspaces using random elements (i.e. spinning random vectors):

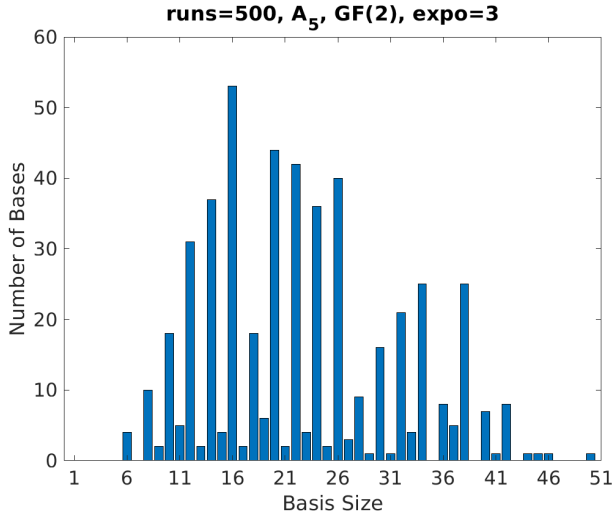


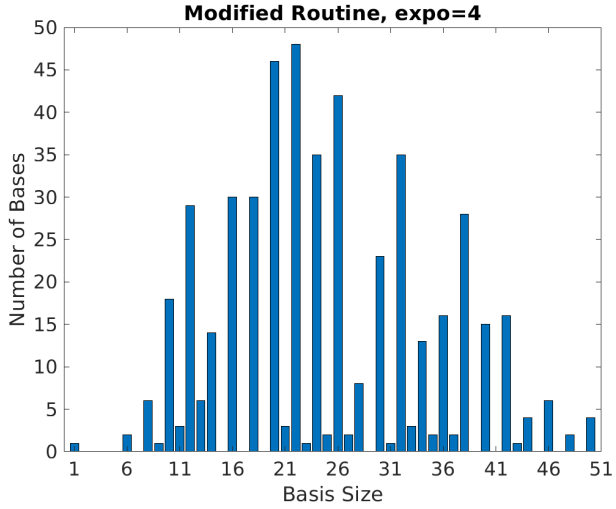
We see that there is a certain bias for subspaces of higher dimension and there is a greater variety of bases generated (in terms of size). Ideally, we would like to control average size of these bases so that we could have greater control of how we were "chopping" our module. Being able to select average subbasis size is precisely the modification this paper seeks to make. How the code functions is as follows:

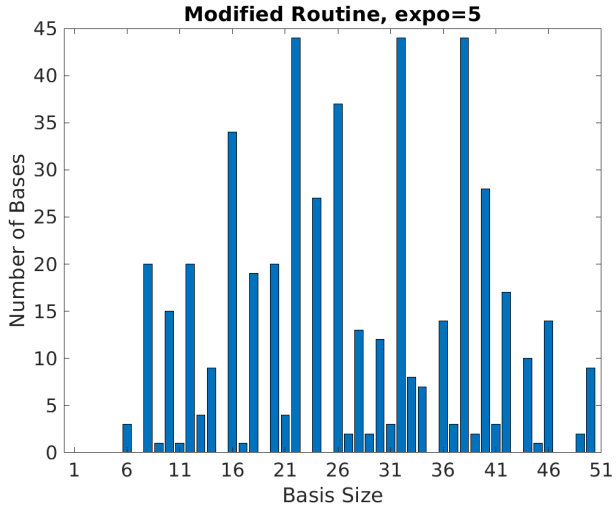
1. A random element of the matrix algebra is selected using Chop's built in word generator. Call this element M.
2. A random vector v is created and the order polynomial of M with respect to v is calculated and factored into irreducibles. Call this order polynomial p(x).
3. The irreducible factor with highest multiplicity in the order polynomial is selected. Call this factor $p_m(x)$ and its multiplicity m.
4. A new polynomial $q(x) = p(x)/(p_m(x))^m$ is created, and finally the program attempts to make the polynomial $q_w(x) = q(x) \cdot (p_m(x))^{(m//expo)}$. Where expo is a number selected by the user and // is integer division.
   4.1. If m//expo = 0 we take $q_w(x) = q(x)$.
   4.2. If m//expo $\neq 0$ we take $q_w(x) = q(x) \cdot (p_m(x))^{(m//expo)}$.
5. The vector $w = q_w(M)(v)$ is created and spun to generate a subbasis.

The process described above is attempting to make a vector in the nullspace of $(p_m(x))^{m-(m//expo)}$. Ideally, such a vector should create a subbasis with dimension significantly greater than m – (m//expo) when spun. Now if the matrix algebra was just generated by M then this vector would create a subbasis of exactly dimension m – (m//expo). The difficulty occurs when w is spun under multiple generators. There is currently no way to predict exactly the size of the space w will create when spun under multiple generators. Instead of rigorously trying to derive bounds for the size of the space w will generate, this paper opts for a numerical exploration of the size of the spaces w creates. The following are graphs generated from using the procedure above with 500 runs and using the regular module of the alternating group A_5 over GF(2):


Modified Routine, expo=2
Number of Bases
Basis Size
0
10
20
30
40
50
60
70
80
1
6
11
16
21
26
31
36
41
46
51


runs=500, A5, GF(2), expo=3
Number of Bases
Basis Size
0
10
20
30
40
50
60
1
6
11
16
21
26
31
36
41
46
51


Modified Routine, expo=4
Number of Bases
Basis Size
0
5
10
15
20
25
30
35
40
45
50
1
6
11
16
21
26
31
36
41
46
51


Modified Routine, expo=5
Number of Bases
Basis Size
0
5
10
15
20
25
30
35
40
45
1
6
11
16
21
26
31
36
41
46
51

With this we see that the average size of bases increases as expo increases. This makes sense because as expo grows larger (m//expo) grows on average smaller so m−(m//expo) grows larger and we end up spinning vectors in a larger nullspace.

4.3 Code

Now we show the actual code used. Note that this code makes use of the CHOP package and is intended to be used in GAP version 4.10.1. We start with the code used to find the vector w as described in step 5 of the ordered list above:

```
NullFinder := function(m, v, expo)
  # Purpose: Given a matrix m and vector v, finds the order polynomial
  # of the matrix m with respect to v and then factors said polynomial,
  # finds the highest degree factor, divdes that factor out of the order
  # polynomial to get a quotient q, rasises the irreduciable factor to its
  # multiplicity in the order polynomial divided by expo (via integer
  # divison), call this polynomial p^(n/expo), and returns the vector
  # w = v*(q*p^(n/expo))(m). Note that w is the smallest vector such
  # that w*p^(n/expo)(m) is zero.
  #
  # Usage: m a matrix, v a vector.
  #
  # Returns: A list with four elements. The first element is a vector w,
  # which is the smallest vector such that w*p^(n/expo)(m) is zero.
  # The second element is p as mentioned above. The third
  # element is the multiplicity of p in the order polynomial.
  # The last element is w*p^(n/expo)(m).

  local vect, mat, ordpol, polyRec, polyFactors, max, highestMult,
  p, q, evalPoly, w, elt, check;

  mat := ShallowCopy(m);
  vect := ShallowCopy(v);
  polyRec := MYCVEC_OrderPoly(mat, vect);
  ordpol := polyRec.ordpol;
  polyFactors := Collected(Factors(ordpol));

  #finding highest degree factor
  max := 0;
  for elt in polyFactors do
    if elt[2] > max then
      max := elt[2];
      highestMult := elt[1];
    fi;
  od;

  #finding w
  p := highestMult^max;
  if Int(max/expo) = 0 then
```

```
    evalPoly := ordpol/p;
    w := HornerEval(evalPoly, mat, vect);
    check := HornerEval(highestMult, mat, w);
  else
    q := ordpol/p;
    evalPoly := q*highestMult^(Int(max/expo)); # max/expo * expo = max
    w := HornerEval(evalPoly, mat, vect);
    check := HornerEval(highestMult^(Int(max/expo)), mat, w);
  fi;

  return [w, highestMult, max, check];
end;
```

This uses a Horner Scheme to compute polynomials of matrices times a vector, which was created by Ari Bormanis and is as follows:

```
HornerEval := function(poly, m, v)
  # Purpose: to use a quicker scheme to evaluate a polynomial
  # of a matrix m and then multiply by a vector v.
  #
  # Usage: poly a polynomial, m a matrix, v a vector.
  #
  # Returns: poly evlautated at m and the multiplied by v.

  local termList, coeffList, highestPow, curVect, curPoly, pow;

  coeffList := CoefficientsOfLaurentPolynomial(poly)[1];
  highestPow := Length(coeffList) - 1;
  curVect := v;
  curPoly := coeffList[1]*v;
  for pow in [1..highestPow] do
     curVect := curVect * m;
     curPoly := curPoly + coeffList[pow + 1]*curVect;
  od;
  return curPoly;
end;
```

Lastly, we have the code that uses the vector w to create a basis for a subspace:

```
ProperSubspaceFast := function(m, expo)
  # INPUT
  # m : module, assuming m is not simple!
  # expo : an integer, NullFinder() will try to take the highest order irreducible
  # factor of the order polynomial, call this p^n, and then return the vector w, which is
  # the smallest vector such that w*p^(n/expo)(M) is zero (where M is the matrix given to
  # NullFinder())
  # OUTPUT
  # sbasis : basis for proper submodule
  # if 500 iterations succeeded prints error message
  # NOTES
```

```
  # 1. generate random word
  # 2. determine good factor of the order polynomial of a
  #    representing matrix from m and the random word
  # 3. evaluate good factor with m and v to obtain a vector w
  # 4. spin up w, if this creates full space return to 1. else
  #    output basis for the space created by spinning up w

  local dim, M, v, subbasis, nullRes, spin, w, i;

  dim := Dimension(m);

  # this part keeps trying random v's to find a proper submodule
  for i in [1..10] do
    NextWord( m!.wg );
    M := EvaluateWord(m!.wg);
    v := M[1];
    if v*M <> Zero(v) then
      subbasis := EmptySemiEchelonBasis(v);
      nullRes := NullFinder(M, v, expo);
      w := nullRes[1];
      CH0P_Spin(RepresentingMatrices(m), subbasis, dim, w);
      if Length(subbasis) < dim then
        return subbasis;
      fi;
    fi;
  od;

  return fail;
end;
```

5 Conclusion

Looking at the results of section 4.2 we see this implementation was largely successful! Especially looking at the graphs with expo = 4 and expo = 5, we see that a large number of subspaces produced were roughly half the size of the space. Furthermore, theoretical predictions for what happens to the bases generated as expo increases were verified. Namely, as expo increases the size of bases produced on average grows larger. This indicates that the modified routine proposed in this paper may be of practical use in speeding up CHOP.

There are many open avenues for further research building off these results. One could feasibly try to derive theoretical bounds for the sizes of bases produced using the modified routine. More interesting still, one could try to assign a probability to the routine producing a basis of some given size for fixed expo. Preliminary investigation into these paths seems to indicate that they might be quite difficult.

Some perhaps more easily obtainable paths forward are as follows: this paper analyzed regular modules, or stated equivalently, regular representations. It would be interesting to see what the new routine produced with other classes of examples like projective indecomposable representations. There is also the question of when, while using the new routine, CHOP should switch back to its old routine. Remember that the modification presented in this paper looks for relatively large modules, but does nothing with proving irreducibility. So at some point it is better to switch back to CHOP's native routine. The question is, what is this point?

Lastly, one could try to achieve the same results of this implementation using what is known as the "condensation method" in computational group theory. It also should be possible to use this new method to put theoretical bounds on the complexity of CHOP.

6 References

1. Neunhöffer, Max. *The MeatAxe*. 2010, `https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.173.6671&rep=rep1&type=pdf`. PowerPoint Presentation.

2. Hyett, Criston. "Optimizations of CHOP: an Extended MeatAxe Algorithm in GAP." Honors Thesis, May 2016.

3. Aluffi, Paolo. *Algebra: Chapter 0*. American Mathematical Society, 2016.

4. Hartley, B., Hawkes, T.O. *Rings, Modules and Linear Algebra*. London: Chapman and Hall, 1970.