A biased overview of computational algebra

Peter Brooksbank

Bucknell University

Linear Algebra and Matrix Theory:
connections, applications and computations

NUI Galway (December 3, 2012)

Lecture Outline

- Lecture 1: Introduction to Computational Algebra.
  - objectives
  - terminology
  - history
  - tools
- Lecture 2: Computing with matrix groups.
  - composition tree
  - recognizing simple groups
  - power of randomization
  - exploiting linear algebra
- Lecture 3: Testing isomorphism of finite groups.
  - automorphisms of p-groups
  - bi-additive maps (bimaps)
  - isometries and pseudo-isometries

What is Computational Algebra?

Computational Algebra seeks efficient algorithms to answer fundamental problems concerning basic algebraic objects (groups, rings, fields etc). Here are some generic examples:

- Given two objects A and B, decide whether $A \cong B$.
- Given A and B, sub-objects of X, compute $A \cap B$.
- Given A sub-object of X, and $x \in X$, decide whether $x \in A$.
- Given A known to be in a classified set of objects, decide which known member A is, and construct an explicit isomorphism.

What is Computational Algebra?

Computational Algebra seeks efficient algorithms to answer fundamental problems concerning basic algebraic objects (groups, rings, fields etc). Here are some generic examples:

- Given two objects A and B, decide whether $A \cong B$.
- Given A and B, sub-objects of X, compute $A \cap B$.
- Given A sub-object of X, and $x \in X$, decide whether $x \in A$.
- Given A known to be in a classified set of objects, decide which known member A is, and construct an explicit isomorphism.

Efficiency may be assessed in a theoretical, or in a practical sense.

In these lectures I will mostly specialize to groups (sometimes rings).

How are groups described?

- Finitely presented groups: given by generators and relations

$$D_8 = \langle x, y \mid x^2 = 1,\ y^4 = 1,\ xy = y^3x \rangle$$

Dehn (1911) *Given a word w in a finitely presented group, decide whether it represents the identity.*

How are groups described?

- Finitely presented groups: given by generators and relations

$$D_8 = \langle x, y \mid x^2 = 1,\ y^4 = 1,\ xy = y^3x \rangle$$

Dehn (1911) *Given a word w in a finitely presented group, decide whether it represents the identity.*

- Permutation groups: given by sets of permutations

$$S_7 = \langle (1\ 2),\ (1\ 2\ 3\ 4\ 5\ 6\ 7) \rangle$$

Rubik's cube *Given an arbitrary configuration of the cube puzzle, find a sequence of moves that solves it.*

How are groups described?

- Finitely presented groups: given by generators and relations

$$D_8 = \langle x, y \mid x^2 = 1,\ y^4 = 1,\ xy = y^3x \rangle$$

Dehn (1911) *Given a word w in a finitely presented group, decide whether it represents the identity.*

- Permutation groups: given by sets of permutations

$$S_7 = \langle (1\ 2),\ (1\ 2\ 3\ 4\ 5\ 6\ 7) \rangle$$

Rubik's cube *Given an arbitrary configuration of the cube puzzle, find a sequence of moves that solves it.*

- Matrix groups: given by sets of invertible matrices over fields

$$\mathrm{SL}(2, \mathbb{Z}/7) = \left\langle \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 6 & 0 \end{bmatrix} \right\rangle$$

Deterministic & Randomized Algorithms

- An algorithm is deterministic if, for any instance of the problem, it terminates with a correct answer in a finite number of steps.
- A randomized algorithm is allowed to make a finite number of random choices (or coin flips) before outputting an answer.
  - An algorithm is Monte Carlo if an upper bound on the chance that it produces an incorrect answer may be prescribed by the user.
  - A Las Vegas algorithm only outputs correct answers, but there is a possibility that it reports `fail`. Again, an upper bound on the likelihood of failure may be prescribed by the user.

Deterministic & Randomized Algorithms

- An algorithm is deterministic if, for any instance of the problem, it terminates with a correct answer in a finite number of steps.
- A randomized algorithm is allowed to make a finite number of random choices (or coin flips) before outputting an answer.
  - An algorithm is Monte Carlo if an upper bound on the chance that it produces an incorrect answer may be prescribed by the user.
  - A Las Vegas algorithm only outputs correct answers, but there is a possibility that it reports `fail`. Again, an upper bound on the likelihood of failure may be prescribed by the user.
- The steps performed by an algorithm depend on the context.
  - Binary operations.
  - Image calculations (permutation groups).
  - Field operations (matrix groups and algebras).

Complexity

The complexity of an algorithm is a measure of the number of steps it takes as a function of input length.

1. If $G = \langle X \rangle$ is a subgroup of the symmetric group S_n, then the input length is $|X|n$.
2. If $G = \langle X \rangle$ is a subgroup of the general linear group $\mathrm{GL}(d, K)$, the length is roughly $|X|d^2$. When $K = \mathbb{F}_q$, this is $|X|d^2 \log q$.

If there is a function f and constant C such that the number of steps carried out by an algorithm for any input of length N at most $Cf(N)$ then we say that it has complexity $O(f(N))$.

Complexity

The complexity of an algorithm is a measure of the number of steps it takes as a function of input length.

1. If $G = \langle X \rangle$ is a subgroup of the symmetric group S_n, then the input length is $|X|n$.
2. If $G = \langle X \rangle$ is a subgroup of the general linear group $\mathrm{GL}(d, K)$, the length is roughly $|X|d^2$. When $K = \mathbb{F}_q$, this is $|X|d^2 \log q$.

If there is a function f and constant C such that the number of steps carried out by an algorithm for any input of length N at most $Cf(N)$ then we say that it has complexity $O(f(N))$.

In Computational Algebra, we care both about

- theoretical complexity (e.g. polynomial time), and
- practical implementations (e.g. in GAP and MAGMA).

Linear Algebra

There are certain linear algebraic problems for which we require efficient solutions.

1. Determine the nullspace of a matrix.
2. Find all solutions of a system of linear equations.
3. Find the product of two matrices.
4. Find and factor the characteristic polynomial of a matrix.
5. Find and factor the minimal polynomial of a matrix.

There are efficient algorithms for all of these problems (e.g. using roughly $d^3 \log^2 q$ basic field operations) and highly optimized computer implementations.

Prehistory

1830 Solubility of polynomials by radicals. (Galois)

1860 Discovery of first sporadic simple groups. (Mathieu)

1911 Formulation of the "Word Problem" for finitely presented groups. (Dehn)

1936 First systematic approach to deciding finiteness of a finitely presented group using "coset enumeration". (Todd & Coxeter)

1951 Suggested use of computational and probabilistic methods to investigate groups of order 256. (Newman)

Early History

1953

- Partial implementation of the Todd-Coxeter algorithm on EDSAC II in Cambridge. (Haselgrove)
- Calculation of characters of symmetric groups on BARK in Stockholm. (Comet)

1959 Subgroup lattices of permutation groups. (Neubüser)

1967 "Computational Problems in Abstract Algebra". (Oxford)

Decade of Discovery

- **Methods**
  1. Management of large permutation groups. (Sims, 1970)
  2. Rewrite systems for f.p. groups. (Knuth-Bendix, 1970)
  3. p-Nilpotent-Quotient method. (Macdonald, 1974)
  4. Reidermeister-Schreier method. (Havas, 1974)
- **Applications**
  1. Existence proof of Lyons' sporadic simple group. (Sims, 1973)
  2. Determination of the Burnside group $B(4,4)$ of order 2^{422}. (Newman & Havas, 1974)
- **Systems**
  1. Aachen-Sydney Group System operational. (1974)
  2. Description of group theory language "Cayley". (Cannon, 1976)

Modern Times

- Existence of the "Baby Monster" of order

$$4,154,781,481,226,426,191,177,580,544,000,000$$

  as a permutation group of degree 13,571,955,000. (Sims)
- Computational techniques used to make and verify the "Atlas of Finite Simple Groups".
- Classification of the 58,760 isomorphism classes of groups of order 2^n, $n \leq 8$. (O'Brien)
- Development of the systems GAP and MAGMA.
- Development of polynomial-time theory for permutation groups.
- Improved methods in computational representation theory.
- Progress in matrix group algorithms.

Specifying Modules

One computes effectively with groups or algebras of $d \times d$ matrices over a field K via their natural underlying module K^d.

- Let A be any K-algebra, which we specify as the enveloping algebra of a set $\{x_1, \ldots, x_r\}$ of generators.

Specifying Modules

One computes effectively with groups or algebras of $d \times d$ matrices over a field K via their natural underlying module K^d.

- Let A be any K-algebra, which we specify as the enveloping algebra of a set $\{x_1, \dots, x_r\}$ of generators.
- A K-vector space M is an A-module if there is a K-algebra homomorphism $\varphi\colon A \to \mathrm{End}_K(M)$.

  Thus, M is specified algorithmically by giving $d \times d$ matrices $a_1, \dots, a_r$ representing the images $\varphi(x_1), \dots, \varphi(x_r)$.

Fundamental Problems

- Homomorphisms. Given A-modules M, N, specified by $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$ respectively, compute (a basis for)

$$\begin{aligned} \mathrm{Hom}_A(M,N) &= \{A\text{-module homomorphisms } M \to N\} \\ &= \{y \colon a_i\, y = y\, b_i \ \ \forall i = 1, \ldots, r\} \end{aligned}$$

Fundamental Problems

- Homomorphisms. Given A-modules M, N, specified by $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$ respectively, compute (a basis for)

$$\begin{aligned}\mathrm{Hom}_A(M,N) &= \{A\text{-module homomorphisms } M \to N\} \\ &= \{y \colon a_i\, y = y\, b_i \ \ \forall i = 1, \ldots, r\}\end{aligned}$$

- Testing Isomorphism. Given A-modules M, N, decide whether or not M and N are isomorphic, and if so find an isomorphism. (i.e. find $g \in \mathrm{GL}(d, \mathbb{F}_q)$ such that $g^{-1} a_i\, g = b_i \ \forall i = 1, \ldots, r$).

Fundamental Problems

- Homomorphisms. Given A-modules M, N, specified by $a_1, \ldots, a_r$ and $b_1, \ldots, b_r$ respectively, compute (a basis for)

$$\begin{aligned} \mathrm{Hom}_A(M,N) &= \{A\text{-module homomorphisms } M \to N\} \\ &= \{y \colon a_i\, y = y\, b_i \ \ \forall i = 1, \ldots, r\} \end{aligned}$$

- Testing Isomorphism. Given A-modules M, N, decide whether or not M and N are isomorphic, and if so find an isomorphism. (i.e. find $g \in \mathrm{GL}(d, \mathbb{F}_q)$ such that $g^{-1} a_i\, g = b_i \ \forall i = 1, \ldots, r$).
- Testing Irreducibility. Given an A-module M, find a proper A-submodule N of M, or else conclude that M is irreducible.

Testing Isomorphism

Theorem (B-Luks (2008))

There is a deterministic, polynomial time algorithm which, given A-modules M and N, decides whether M and N are isomorphic.

Main Idea:

1. If $M \cong N$, then $\mathrm{Hom}_A(M, N) \cdot \mathrm{Hom}_A(N, M) \subset \mathrm{End}_A(M)$ is not nilpotent. Let $d = \dim M = \dim N$.
2. Find $x \in \mathrm{Hom}_A(M, N)$ and $y \in \mathrm{Hom}_A(N, M)$ such that $b = x \cdot y$ is not nilpotent, and put $c = y \cdot x$.
3. Write $M = M\, b^d \oplus \ker b^d$ and $N = N\, c^d \oplus \ker c^d$. Then x induces an isomorphism $M\, b^d \to N\, c^d$.
4. Recursively test isomorphism of $\ker b^d$ and $\ker c^d$.

Generating Random Elements

To make randomized algorithms useful, one first needs an effective method of generating random elements in groups and algebras.

- Algebras. Compute a K-basis for the algebra, and then take a random linear combination of basis elements.

Generating Random Elements

To make randomized algorithms useful, one first needs an effective method of generating random elements in groups and algebras.

- Algebras. Compute a K-basis for the algebra, and then take a random linear combination of basis elements.
- Groups. Two results on random generation in finite groups:
    - Babai (1991) The first polynomial time algorithm to construct independent, (nearly) uniformly distributed random elements.
    - CLMNO (1995) Described the "product replacement" algorithm as a practical alternative. With easy modifications, can also be used to generate "random" elements of algebras.

Testing Irreducibility (The Meataxe)

Given: An A-module M by generators $a_1, \dots, a_r$.
Find: A proper A-submodule of M (or decide that no such exists).

1. Choose a random θ in $\mathrm{Env}(a_1, \dots, a_r)$.
2. Compute and factor the minimal polynomial of θ.
3. For an irreducible factor π, compute nullspace N of $\xi = \pi(\theta)$.
4. If $0 \neq v \in N$ generates a proper A-submodule, return it.
5. Else, if $\deg(\pi) = \dim(N)$, compute N^*, the nullspace of ξ^{tr}.
   - If $0 \neq w \in N^*$ generates a proper submodule W of M^*:
     - Choose any $0 \neq u$ of $W^\perp$.
     - Return the A-submodule generated by u.
   - Else report that M is irreducible.
6. Repeat steps 3-5 for a new irreducible factor, or return to step 1.

Correctness of The Meataxe

The only output that is in question is the report "M is irreducible" which occurs only when $\deg(\pi) = \dim(N)$ in step 5.

Suppose, in that case, that M has a proper submodule L. Observe:

1. $\theta|_N$ is irreducible with minimal polynomial π.
2. $L \cap N$ is either 0 or N.
   2.1 If $L \cap N = N$, any $v \in N$ lies in L: a proper submodule is found.
   2.2 If $L \cap N = 0$, let $e_1, \ldots, e_c, e_{c+1}, \ldots, e_d$ exhibit L. Then

$$\xi = \begin{bmatrix} A & \cdot \\ B & C \end{bmatrix} \text{ and } \xi^{\text{tr}} = \begin{bmatrix} A^{\text{tr}} & B^{\text{tr}} \\ \cdot & C^{\text{tr}} \end{bmatrix}.$$

- Now, A has maximal rank, so that nullity ξ^{tr} = nullity C^{tr}.
- Hence any vector in the nullspace of ξ^{tr} also lies in a proper submodule of the dual module M^* (defined by ${a_1}^{\text{tr}}, \ldots, {a_r}^{\text{tr}}$).
- But then any vector in the orthogonal complement of such a submodule must lie in a proper submodule of M.