Computer Algebra
Lecture 1

James Davenport

University of Bath

3 September 2018

Computer Algebra (Symbolic Computation)

"Getting computers to do algebra"

computers by themselves can't even do arithmetic: there are only finitely many `int` in C for example. And $(1+10^{20})-10^{20} \stackrel{\texttt{double}}{\longrightarrow} 0$ whereas $1+(10^{20}-10^{20}) \stackrel{\texttt{double}}{\longrightarrow} 1$.

to do Do we really know algorithms? Can you factor $x^5-2x^4+8x^3+3x^2+6x-4$?

algebra and much geometry can be turned into algebra. So can much of calculus.

Some early history

[But I don't know anything about Chinese developments]

- 1948 Manchester Baby is first stored-program computer.
- 1951 (Cambridge, UK) $180\left(2^{127}-1\right)^2+1$ is prime [MW51].
- 1952 Elliptic curve calculations on MANIAC [ST92, p. 119].
- 1953 two US theses [Kah53, Nol53] kicked off the 'calculus' side of computer algebra with programs to differentiate expressions.
- 1953 (Cambridge, UK) a group theory algorithm was implemented [Has53].

Thereafter, history rather splits into "polynomial/calculus" and "group theory".

“Polynomial/Calculus” systems

[Kah53, Nol53] could only do one thing, but soon people wrote systems, common by early 1970s.

SAC Written in Fortran [Col71], its descendant QEPCAD [Bro03] alive today

MACSYMA in LISP at MIT [MF71, PW85], available (as Maxima) in SAGE

Reduce in LISP at Utah [Hea71, Hea05] (our system)

SCRATCHPAD in LISP at IBM [GJY75], now dead, but inspired Axiom [JS92].

COBALG in COBOL at Cambridge [fHN76], now dead.

Maple Kernel in C, Waterloo (CA), 1983– [CGGG83].

Mathematica Kernel in C, Wolfram Research, 1988– [Wol88].

Are these some of the longest-lived programs?

Group theory programs

I know less about these.

CAYLEY [BC90] was a major group theory system, replaced by

MAGMA [BCM94] a system built on Axiom-like ideas.

GAP [BL98], design inspired by Maple. Available in SAGE.

Why two different directions? Polynomial systems are concerned with a *specific* polynomial, whereas group theory systems are concerned with a *specific* group of permutations, rather than a specific permutation.

Algebraic geometry systems, such as SINGULAR [Sch03] (available in SAGE), Macauley [BS86] or COCOA [GN90], are concerned with *specific* sets of polynomials.

Reduce

This is the first system I want to consider. It is relatively simple, but still current (last year won its division of the Satisfiability Modulo Theories competition `http://smtcomp.sourceforge.net/2017/results-NRA.shtml`).
It is free, and has a free manual [HS18].
Written in LISP, which means it's simple enough to understand.
This afternoon, you'll install and run it on your computers, but now I will demonstrate it.

Storage of Polynomials

How do we store polynomials

$$p_{\text{dense}} = \sum_{i=0}^{n} a_i x^i? \tag{1}$$

Your first thought might be a vector $[a_0, a_1, \ldots, a_n]$ of all the coefficients. Such a method is called *dense*.
However, for $x^{1000000} + 1$ is would take megabytes of memory to store, which seems silly. After all, we didn't write down all those zeros, so why should a computer? Instead consider

$$p_{\text{sparse}} = \sum_{i=1}^{m} b_i x^{e_i} : b_i \neq 0 \tag{2}$$

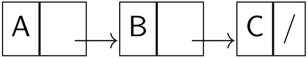
and store the pairs (b_i, e_i).
Such a storage method is sparse.

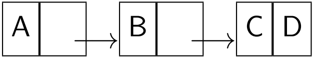
LISP [McC60]

One of the oldest programming language, the fundamental data structure is the CONS cell, composed of two parts, known for historical reasons as CAR and CDR (though many people think of them as `first` and `rest`, which works if you have a list).

If we have such an object 

, we write `(A . B)`.

A list 

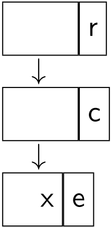
(where the last CDR points to a special object NIL), which should be `(A. (B. (C. NIL)))` is written `(A B C)`. If we had 

we would write `(A B C . D)`.

Also, instead of `CAR(CDR(CDR(Z)))` (the third element of the list `Z`), we write `CADDR(Z)`.

I could say a lot more about LISP, but don't need to.

If we have a non-trivial (i.e. not just a number) polynomial p, we can write it as $p = cx^e + r$, with $e \neq 0$, e.g. $3x^2 + 5$ or $\underbrace{(3y+2)}_{c}x^7 + \underbrace{(2y^2-4)x^6 + 6y^4 + 7}_{r}$. c is a *non-zero* polynomial *not* involving x, and r might involve x, but only to a power $< e$. r is called the *reductum*, c the *leading coefficient*, x the *main variable* and e the *exponent*.



or `(((x . e) . c) . r)`

x is the CAAAR (or `mvar`)
e is the CDAAR (or `ldeg`)
c is the CDAR (or `lc`)
r is the CDR (or `red`)

Reduce implementation in LISP continued

Of course, not everything is a polynomial.

But we can view everything as a fraction of two polynomials $\frac{\text{numerator}}{\text{denominator}}$, known as a *standard quotient* (SQ).

This is easy: | numerator | denominator |.

Once Reduce has simplified something to being an SQ, it likes to keep it like that. Hence a standard quotient z is stored as `(!*SQ z NIL)`, or `(!*SQ z T)` if the system knows it has been simplified.

```
69: z;
```

$$x + y$$

```
70: lisp get('z,'avalue);

(scalar (!*sq ((((x . 1) . 1) ((y . 1) . 1)) . 1) nil))
```

Reduce

```
73: sub(n=1023,f*g);
```

$$a^b x^{1023} + a^b y^{1023} + b^{1023} x^{1023} + b^{1023} y^{1023} + c^{1023} x^{1023} + c^{1023} y^{1023}$$

```
74: f:=a^n+b^n+c^n;
```

$$f := a^n + b^n + c^n$$

```
75: g:=x^n+y^n;
```

$$g := x^n + y^n$$

```
76: sub(n=1023,f*g);
```

$$a^{1023} x^{1023} + a^{1023} y^{1023} + b^{1023} x^{1023} + b^{1023} y^{1023} + c^{1023} x^{1023} + c^{1023} y^{1023}$$

```
77: lisp ws;

(!*sq ((((a . 1023) ((x . 1023) . 1) ((y . 1023) . 1)) ((b . 1023) ((x . 1
1) ((y . 1023) . 1)) ((c . 1023) ((x . 1023) . 1) ((y . 1023) . 1))) . 1)
```

Maple's poly format [MP14]

$f := a^n + b^n + c^n;$

$$f := a^n + b^n + c^n$$

$g := x^n + y^n;$

$$g := x^n + y^n$$

$dismantle(expand(eval(f \cdot g, n = 1023)));$

```
POLY(14)
   EXPSEQ(6)
      NAME(4): a
      NAME(4): b
      NAME(4): c
      NAME(4): x
      NAME(4): y
   DEGREES(HW): ^2046 ^1023 ^0 ^0 ^1023 ^0
   INTPOS(2): 1
   DEGREES(HW): ^2046 ^1023 ^0 ^0 ^0 ^1023
   INTPOS(2): 1
   DEGREES(HW): ^2046 ^0 ^1023 ^0 ^1023 ^0
   INTPOS(2): 1
   DEGREES(HW): ^2046 ^0 ^1023 ^0 ^0 ^1023
   INTPOS(2): 1
```

Maple's sum format

$dismantle(expand(eval(f \cdot g, n=1024)));$

```
SUM(13)
   PROD(5)
      NAME(4): c
      INTPOS(2): 1024
      NAME(4): y
      INTPOS(2): 1024
   INTPOS(2): 1
   PROD(5)
      NAME(4): c
      INTPOS(2): 1024
      NAME(4): x
      INTPOS(2): 1024
   INTPOS(2): 1
   PROD(5)
      NAME(4): b
      INTPOS(2): 1024
      NAME(4): y
      INTPOS(2): 1024
   INTPOS(2): 1
```

Taxonomy of representations

Reduce's representation is

- **sparse** (only non-zero c are stored)
- **recursive** (polynomials in x whose coefficients are polynomials in the rest)

Reduce *prints* distributed, but stores recursive

- **sparse in variables** (we don't store any intermediate y^0)

Maple's representation is

- **sparse** (only non-zero c are stored)
- **distributed** (multiplied out)

sum **sparse in variables** (we don't store any intermediate y^0)

poly **dense in variables** (y^0 stored)

But all of them require some order on the variables to know which is "main".

How big are they?

$$\underbrace{(a^n + b^n + c^n)}_{f} * \underbrace{(x^n + y^n)}_{g}$$

Category	Maple `sum`	Maple `poly`	Reduce
Variables	12	5	< 9
Exponents	12	(6)	< 9
Coefficients	6	6	6
Data Structure	13+6*5=43	14+6=20	$< 2 * 27 = 54$
Generalised	$7t_f t_g + 1$	$2t_f t_g + t_f + t_g + 3$	$< 6t_f(t_g + 1)$

We can see why [MP14] invented `poly`!

The $<$ for Reduce is because of sharing of CONS cells.

Bibliography I

- G. Butler and J. Cannon.
  The Design of Cayley — A Language for Modern Algebra.
  In *Proceedings DISCO '90*, 1990.
- W. Bosma, J. Cannon, and G. Matthews.
  Programming with algebraic structures: design of the Magma language.
  In *Proceedings ISSAC 1994*, pages 52–57, 1994.
- T. Breuer and S.A. Linton.
  The GAP4 Type System Organising Algebraic Algorithms.
  In O. Gloor, editor, *Proceedings ISSAC '98*, pages 38–45, 1998.
- C.W. Brown.
  QEPCAD B: a program for computing with semi-algebraic sets using CADs.
  ACM SIGSAM Bulletin 4, 37:97–108, 2003.

Bibliography II

D. Bayer and M. Stillman.
The Design of Macaulay: A System for Computing in Algebraic Geometry and Commutative Algebra.
In *Proceedings SYMSAC 86*, pages 157–162, 1986.

B.W. Char, K.O. Geddes, M.W. Gentleman, and G.H. Gonnet.
The Design of MAPLE: A Compact, Portable and Powerful Computer Algebra System.
In *Proceedings EUROCAL 83*, pages 101–115, 1983.

G.E. Collins.
The SAC-1 System: An Introduction and Survey.
In *Proceedings SYMSAC 1971*, 1971.

Bibliography III

J.P. ffitch, P. Herbert, and A.C. Norman.
Design Features of COBALG.
In R.D. Jenks, editor, *Proceedings SYMSAC 76*, pages 185–188, 1976.

J.H. Griesmer, R.D. Jenks, and D.Y.Y. Yun.
SCRATCHPAD User's Manual.
IBM Research Publication RA70, 1975.

A. Giovini and G. Niesi.
CoCoA: A User-Friendly System for Commutative Algebra.
In *Proceedings DISCO '90*, 1990.

C.B. Haselgrove.
Implementations of the Todd-Coxeter Algorithm on EDSAC-1.
Unpublished, 1953.

- A.C. Hearn.
  REDUCE 2: A system and language for algebraic manipulation.
  In *Proceedings of the second ACM symposium on Symbolic and algebraic manipulation*, pages 128–133, 1971.
- A.C. Hearn.
  REDUCE: The First Forty Years.
  In T. Sturm A. Dolzmann, A. Seidl, editor, *Proceedings A3L*, pages 19–24. Books on Demand GmbH, 2005.
  URL: `reduce-algebra.com/reduce40.pdf`.
- A.C. Hearn and R. Schöpf.
  REDUCE User's Manual (Free Version; June 8, 2018).
  `http://reduce-algebra.sourceforge.net/`, 2018.

Bibliography V

R.D. Jenks and R.S. Sutor.
AXIOM: The Scientific Computation System.
Springer-Verlag, 1992.

H.G. Kahrimanian.
Analytic differentiation by a digital computer.
Master's thesis, Temple U Philadelphia, 1953.

J. McCarthy.
Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I.
Comm. ACM, 3:184–195, 1960.

W.A. Martin and R.J. Fateman.
The MACSYMA System.
In *Proceedings Second Symposium on Symbolic and Algebraic Manipulation*, pages 59–75, 1971.

Bibliography VI

- M. Monagan and R. Pearce.
  POLY : A new polynomial data structure for Maple 17.
  In R. Feng *et al.*, editor, *Proceedings Computer Mathematics*, pages 325–348, 2014.
- J.C.P. Miller and D.J. Wheeler.
  Large Prime Numbers.
  Nature, 168:838, 1951.
- J. Nolan.
  Analytic differentiation on a digital computer.
  Master's thesis, Math. Dept. M.I.T., 1953.
- R. Pavelle and P.S. Wang.
  MACSYMA from F to G.
  J. Symbolic Comp., 1:69–100, 1985.

Bibliography VII

H. Schönemann.
Singular in a Framework for Polynomial Computations.
Algebra Geometry and Software Systems, pages 163–176, 2003.

J.H. Silverman and J. Tate.
Rational Points on Elliptic Curves.
Springer-Verlag, 1992.

S. Wolfram.
Mathematica: A System for Doing Mathematics by Computer.
Addison-Wesley, 1988.