Methodologies of Symbolic Computation

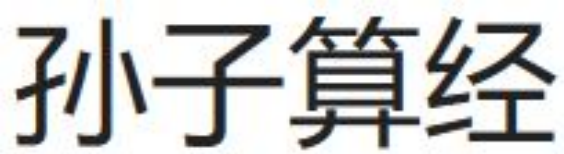
James Davenport

University of Bath

18–19 September 2018

Structure

1. Introduction and History
2. Better straightforward algorithms
3. Modular (Chinese Remainder) algorithms



4. Hensel (p-adic) algorithms
5. Not so straightforward algorithms
6. How the subject works
7. Convergence with AI?

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 $(1+10^{20})-10^{20} \overset{\texttt{double}}{\longrightarrow} 0$ whereas $1+(10^{20}-10^{20}) \overset{\texttt{double}}{\longrightarrow} 1$.

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.

The algorithmic need to cover all cases

We are normally taught to solve $ax^2 + bx + c$ as

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

but in fact the true answer is

$$\begin{cases} a \neq 0 & \Rightarrow \quad x = \frac{-b \pm \sqrt{b^2-4ac}}{2a} \\ a = 0, b \neq 0 & \Rightarrow \quad x = -c/b \\ a = b = 0, c \neq 0 & \Rightarrow \quad \text{contradiction} \\ a = b = c = 0 & \Rightarrow \quad x = \text{anything} \end{cases}$$

Note that the number of solutions isn't necessarily 2.

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].
- 1958 McCarthy invents LISP [McC60], used for SHRDLU and Macsyma (and many others).
  - * These two fields have a common ancestry.

How did they split?

Probably best illustrated by looking at integration.

1961 SAINT: heuristic integration "better than a freshman" [Sla61, Harvard]

1967 SIN: algorithmic integration "algorithmically complete" [Mos67, M.I.T]

However, "algorithms are better" has its limitations, see [JR10]: if you want answers such as a human would generate, then you want to build on human reasoning.

Straightforward Algorithms: Fractions are Expensive

$$\begin{aligned} a(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5; \\ b(x) &= 3x^6 + 5x^4 - 4x^2 - 9x - 21. \end{aligned}$$

$$b_1 = \frac{-5}{9}x^4 + \frac{127}{9}x^2 - \frac{29}{3},$$

$$b_2 = \frac{50157}{25}x^2 - 9x - \frac{35847}{25}$$

$$\begin{aligned} b_3 &= \frac{93060801700}{1557792607653}x + \frac{23315940650}{173088067517} \\ b_4 &= \frac{761030000733847895048691}{86603128130467228900}. \end{aligned}$$

And they'd be really expensive if we had other variables around, as we'd have to do g.c.d. calculations to cancel the fractions, or they would grow greatly.

Just fraction-free is also expensive

$$
\begin{aligned}
a(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5; \\
b(x) &= 3x^6 + 5x^4 - 4x^2 - 9x - 21. \\
b_1 &= -15\,x^4 + 381\,x^2 - 261 \\
b_2 &= 6771195\,x^2 - 30375\,x - 4839345 \\
b_3 &= 5007452958520282125 00\,x + 1129134141014747231250 \\
b_4 &= 74366224225404865381141772558558905729564453125 00
\end{aligned}
$$

But any old fool can see that there are common factors!

So remove common factors

$$
\begin{aligned}
a(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5; \\
b(x) &= 3x^6 + 5x^4 - 4x^2 - 9x - 21. \\
b_1 &= -5\,x^4 + 127\,x^2 - 87 \qquad \text{Cancelled } 3 \\
b_2 &= 5573\,x^2 - 25\,x - 3983 \qquad \text{Cancelled } 1215 = 3^5 \cdot 5 \\
b_3 &= 1861216034\,x + 4196869317 \qquad \text{Cancelled } 3^{16} \cdot 5^5 \cdot 2 \\
b_4 &= 1
\end{aligned}
$$

This is in fact a perfectly reasonable algorithm for $\mathbf{Z}[x]$, and, if I didn't know better (see later) is the one I would use for $\mathbf{Z}[x]$.
But all those pp are a great many g.c.d. in R, and if $R = S[y]$, many more computations over S, and if $S = T[z]$ ...
All those cancellations (apart from the 2) were of leading coefficients. It turns out we can predict these.

Subresultant Polynomial Remainder Seq [Bro71a, Bro71b]

1: **procedure** $\mathrm{SREuclid}(f, g)$ ▷ Almost the g.c.d. of $a, b \in R[x]$
2: **if** $\deg(f) < \deg(g)$ **then**
3: $a_0 \leftarrow \mathrm{pp}(g)$; $a_1 \leftarrow \mathrm{pp}(f)$;
4: **else**
5: $a_0 \leftarrow \mathrm{pp}(f)$; $a_1 \leftarrow \mathrm{pp}(g)$;
6: **end if**
7: $\delta_0 \leftarrow \deg(a_0) - \deg(a_1)$;
8: $\beta_2 \leftarrow (-1)^{\delta_0+1}$; $\psi_2 \leftarrow -1$; $i \leftarrow 1$;
9: **while** $a_i \neq 0$ **do**
10: $a_{i+1} = \mathrm{prem}(a_{i-1}, a_i)/\beta_{i+1}$;
11: $\delta_i \leftarrow \deg(a_i) - \deg(a_{i+1})$; $i \leftarrow i + 1$;
12: $\psi_{i+1} \leftarrow (-\mathrm{lc}(a_{i-1}))^{\delta_{i-2}} \psi_i^{1-\delta_{i-2}}$;
13: $\beta_{i+1} \leftarrow -\mathrm{lc}(a_{i-1}) \psi_{i+1}^{\delta_{i-1}}$;
14: **end while**
15: **return** $\mathrm{pp}(a_{i-1})$ ▷ The gcd is this, up to a factor in R
16: **end procedure**

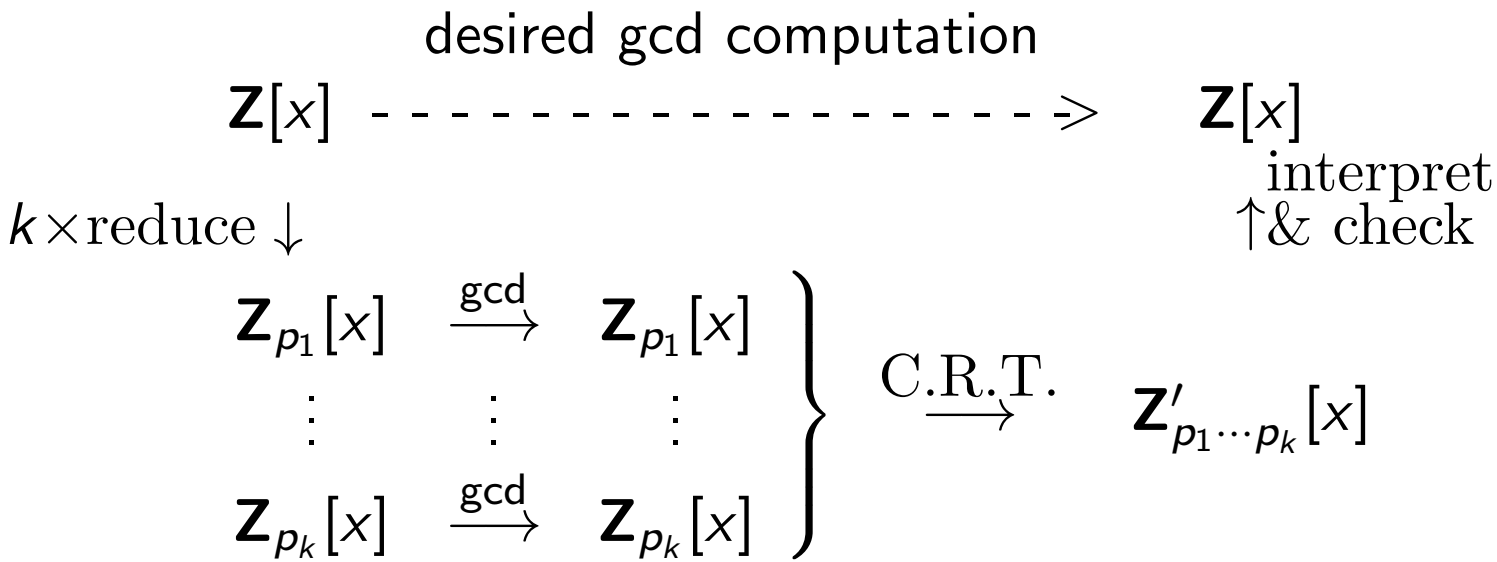
Trivial g.c.d. modulo 5

Write P_5 to signify the polynomial P considered as a polynomial with coefficients modulo 5. $P = \gcd(A, B)$ implies that P_5 divides A_5. Similarly, P_5 divides B_5, and therefore it is a common divisor of A_5 and B_5.
But calculating the g.c.d. of A_5 and B_5 is fairly easy:

$$\begin{aligned}
A_5(x) &= x^8 + x^6 + 2x^4 + 2x^3 + 3x^2 + 2x; \\
B_5(x) &= 3x^6 + x^2 + x + 1; \\
C_5(x) &= \mathrm{rem}(A_5(x), B_5(x)) = A_5(x) + 3(x^2 + 1)B_5(x) = 4x^2 + 3; \\
D_5(x) &= \mathrm{rem}(B_5(x), C_5(x)) = B_5(x) + (x^4 + 4x^2 + 3)C_5(x) = x; \\
E_5(x) &= \mathrm{rem}(C_5(x), D_5(x)) = C_5(x) + xD_5(x) = 3.
\end{aligned}$$

Thus A_5 and B_5 are relatively prime, which implies that $P_5 = 1$. As the leading coefficient of P has to be one, we deduce that $P = 1$.

Diagrammatic illustration of Modular GCD Algorithm



$\mathbf{Z}'_{p_1\cdots p_k}[x]$ indicates that some of the p_i may have been rejected by the compatibility checks, so the product is over a subset of $p_1 \cdots p_k$.

More generally

gcd could be almost any algorithm that works over the integers. But we always have these questions.

1. Are there "good" reductions from R?
   * A weak answer is that the algorithm over the integers can only test finitely many numbers for $= 0$, so avoiding their prime factors is sufficient
2. How can we tell if R_i is good?
3. How many reductions should we take?
4. How do we combine?
   * Generally Chinese Remainder Theorem, with Farey reconstruction [WGD82] if we want rationals. **But** we need to know what to combine.
5. How do we check the result?

Generally very problem-dependent

Polynomial Factorisation

- At the beginning, I asked "Do we really know algorithms?
- Can you factor $f := x^5 - 2x^4 + 8x^3 + 3x^2 + 6x - 4$?"
- Well, of course
  $f \cdot (x-1) = x^6 - 3x^5 + 10x^5 - 5x^3 + 3x^2 - 10x + 4 =$
  $(x^3 - 1) * (x^3 - 3x^2 + 10x - 4)$,
- so $f = (x^2 + x + 1) * (x^3 - 3x^2 + 10x - 4)$,
- and it's not hard to see that both factors are irreducible.
- Hardly an algorithm!
- Can't we work modulo primes p?
- And in fact there are good algorithms for polynomial factorisation modulo p.

So onward to modular methods?

The questions

1. Are there "good" reductions from R?

factor$_{p_i}$ is not the image of some factor$_{\mathbf{Z}}$ so the "avoid finitely many divide by 0" argument doesn't work.

2. How can we tell if R_i is good?

* Not obvious

3. How many reductions should we take?

* As for g.c.d. (Landau–Mignotte)?

4. How do we combine?

Which factor modulo p_1 belongs with which factor modulo p_2 belongs with which factor modulo p_3 . . . ?

5. How do we check the result?

Not obvious, especially assertions of irreducibility.

These are real issues

- x^4+1 is irreducible, but factors as two quadratics modulo every prime
- Not unique: "Swinnerton-Dyer polynomials" [SD70]
- x^4+3 factors as

$$x^4+3 = \left(x^2+2\right)\left(x+4\right)\left(x+3\right) \mod 7; \quad (1)$$

$$x^4+3 = \left(x^2+x+6\right)\left(x^2+10\,x+6\right) \mod 11. \quad (2)$$

So really

$$x^4+3 = \left(x^2+2\right)\left(x^2+5\right) \mod 7, \quad (3)$$

Do we pair $\left(x^2+x+6\right)$ with $\left(x^2+2\right)$ or $\left(x^2+5\right)$? Both seem feasible, and both are correct. Modulo 77, we do not have unique factorisation.

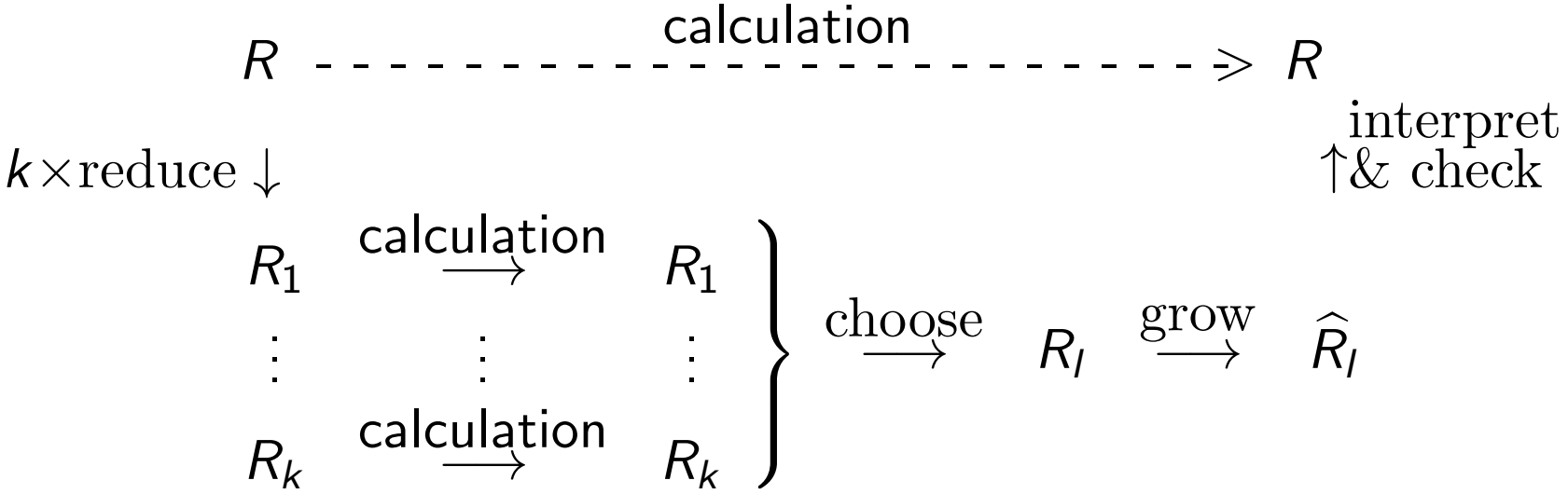
We need a different idea

Write f_p to mean $f \pmod p$ etc., and $f_{p^2} = f_p + pf_2$ etc.
Suppose we are factoring f as $f = gh$ and we know g_p, h_p such that $f_p = g_p h_p \pmod p$. Then
$f_{p^2} = g_{p^2} h_{p^2} = g_p h_p + p(g_p h_2 + h_p g_2) + p^2 g_2 h_2$. Then

$$\frac{f_{p^2} - g_p h_p}{p} \equiv g_p h_2 + h_p g_2 \pmod p \tag{4}$$

(4) is *linear* in the unknowns g_2 and h_2, and can be solved efficiently for unique g_2, h_2 as long as their degrees are less than g_p, h_p. Similarly we can solve from $f_{p^3} = g_{p^3} h_{p^3}$ to get a linear equation in the unknowns g_3 and h_3, and so on.
Eventually we know g_{p^n}, h_{p^n} such that $p^n > 2LM$, where LM is the Landau–Mignotte bound on factors of f, and so g_{p^n} should be g etc.
This process is known as *Hensel Lifting* and lets us go from a solution modulo p to one modulo p^n

Diagrammatic illustration of Hensel Algorithms



where R_l is one of $R_1, \ldots, R_k$.

Not straightforward (I): Gröbner

Gröbner bases [Buc65] can be seen as doing for nonlinear equations what Gaussian elimination does for linear equations. But there are genuine complications.

1. $\{x^2-1, y^2-1, (x-1)(y-1)\}$ defines $(-1,1),(1,-1),(1,1)$ as three points, but it takes three equations in two unknowns to define precisely that.
2. $\{(x+1-y)(x-6+y),(x+1-y)(y-3)\}$ defines $x=y-1$ (a straight line) *and* the point $(3,3)$.

Complexity can be doubly-exponential in the number of variables [Yap91, Chi09]. Nevertheless, these are a major algorithmic step forward.

Not straightforward (II): Integration

[Mos67] was based on theory by Risch [Ris69], making algorithmic ideas of Liouville [Lio35].
The key idea can be seen in "e^{-x^2} has no integral".

Analysis Clearly there is: e^{-x^2} is continuous

Algebra There is no *formula* that differentiates to e^{-x^2}

* Saying "erf" is cheating because that's the definition of erf.

Hence computers implementing this algorithm can do what 99% of humans, probably even 99% of mathematicians, cannot, and *prove* things unintegrable.
Or find some quite surprising results [Dav86]:

$$\int \exp\left(\frac{1}{e^x+1}-10x\right)\frac{2{,}581{,}284{,}541e^x+1{,}757{,}211{,}400}{39{,}916{,}800e^{3x}+119{,}750{,}400e^{2x}+119{,}750{,}400e^x+39{,}916{,}800} =$$

$$\exp\left(\frac{1}{e^x+1}-10x\right)\frac{39{,}916{,}800e^{11x}+19{,}958{,}400e^{9x}-26{,}611{,}200e^{8x}+\cdots-175{,}721{,}140}{39{,}916{,}800e^x+39{,}916{,}800}.$$

Methodology of Computer Algebra

There is little doubt that, in the minds of most researchers, the ideal computer algebra paper consists of a problem statement, a new algorithm, a complexity analysis and a few validating examples.
However, these complexity results tend to be in the dense setting, while most practical work is done in the sparse setting. In that setting many of these problems are NP-hard (even g.c.d. on univariate polynomials [Pla78]) or have even doubly-exponential worst-case bounds [Yap91, DH88].
The SAT community handles its NP-completeness by annual contests and large suites of benchmarks, which computer algebra has hitherto been weak at.

Is it time for SC to look again at AI?

SAT handles NP-completeness far better than SC does.

Also will generally try several approaches to a problem, whereas algebra systems tend not to

This makes algebra systems much less suitable as black boxes

Order (of variables, say) is inherent when we come to compute, though not in the problem statement. This can really matter [BD07, DSS04].

Heuristics to choose these and other parameters are under-researched (but see [Eng18]).

What is SC's equivalent of the Hammer methodology?

Or What *should* SC's equivalent of the Hammer methodology be?

Example (Dense g.c.d.s 1: [Sch03])

$$\gcd(\underbrace{x^{pq}-1}_{f(x)},\underbrace{x^{p+q}-x^p-x^q+1}_{g(x)}) = \frac{(x^p-1)(x^q-1)}{x-1} = \underbrace{x^{p+q-1}-x^{p+q-2}\pm\cdots-1}_{h(x)}. \tag{5}$$

Example (Dense g.c.d.s 2)

Therefore

$$\gcd(f(x_1)f(x_2)\cdots f(x_k), g(x_1)g(x_2)\cdots g(x_k)) = h(x_1)\cdots h(x_k), \tag{6}$$

and the righthand side has $(2\min(p,q))^k$ terms, whereas the arguments to gcd have 2^k and 4^k terms.

Bibliography I


C.W. Brown and J.H. Davenport.
The Complexity of Quantifier Elimination and Cylindrical Algebraic Decomposition.
In C.W. Brown, editor, *Proceedings ISSAC 2007*, pages 54–60, 2007.


E.R. Berlekamp.
Factoring Polynomials over Large Finite Fields.
Math. Comp., 24:713–735, 1970.


W.S. Brown.
On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors.
In *Proceedings SYMSAC 1971*, pages 195–211, 1971.

Bibliography II

W.S. Brown.
On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors.
J. ACM, 18:478–504, 1971.

B. Buchberger.
Ein Algorithmus zum Auffinden des Basiselemente des Restklassenringes nach einem nulldimensionalen Polynomideal.
PhD thesis, Math. Inst. University of Innsbruck, 1965.

A.L. Chistov.
Double-exponential lower bound for the degree of any system of generators of a polynomial prime ideal.
St. Petersburg Math. J., 20:983–1001, 2009.

Bibliography III


J.H. Davenport.
On the Risch Differential Equation Problem.
SIAM J. Comp., 15:903–918, 1986.


J.H. Davenport and J. Heintz.
Real Quantifier Elimination is Doubly Exponential.
J. Symbolic Comp., 5:29–35, 1988.


A. Dolzmann, A. Seidl, and Th. Sturm.
Efficient Projection Orders for CAD.
In J. Gutierrez, editor, *Proceedings ISSAC 2004*, pages 111–118, 2004.

Bibliography IV


M. England.
Machine Learning for Mathematical Software.
In J.H. Davenport, M. Kauers, G. Labahn, and J. Urban, editors, *Proceedings Mathematical Software — ICMS 2018*, pages 165–174, 2018.


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


D.J. Jeffrey and A.D. Rich.
Reducing Expression Size Using Rule-Based Integration.
In S. Autexier *et al.*, editor, *Proceedings CICM 2010*, pages 234–246, 2010.

Bibliography V


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


J. Liouville.
Mémoire sur l'intégration d'une classe de fonctions transcendantes.
Crelle's J., 13:93–118, 1835.


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


J. Moses.
Symbolic Integration.
PhD thesis, M.I.T. & Project MAC TR-47, 1967.

Bibliography VI


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.


D.A. Plaisted.
Some Polynomial and Integer Divisibility Problems are *NP*-Hard.
SIAM J. Comp., 7:458–464, 1978.


R.H. Risch.
The Problem of Integration in Finite Terms.
Trans. A.M.S., 139:167–189, 1969.

Bibliography VII


A. Schinzel.
On the greatest common divisor of two univariate polynomials, I.
In *A Panorama of number theory or the view from Baker's garden*, pages 337–352. C.U.P., 2003.


H.P.F. Swinnerton-Dyer.
Letter to E.H. Berlekamp.
Mentioned in [Ber70], 1970.


J. Slagle.
A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus.
PhD thesis, Harvard U., 1961.

Bibliography VIII


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


P.S. Wang, M.J.T. Guy, and J.H. Davenport.
p-adic Reconstruction of Rational Numbers.
SIGSAM Bulletin, 16(2):2–3, 1982.


C.K. Yap.
A new lower bound construction for commutative Thue systems with applications.
J. Symbolic Comp., 12:1–27, 1991.