Computer Algebra
Lecture 3

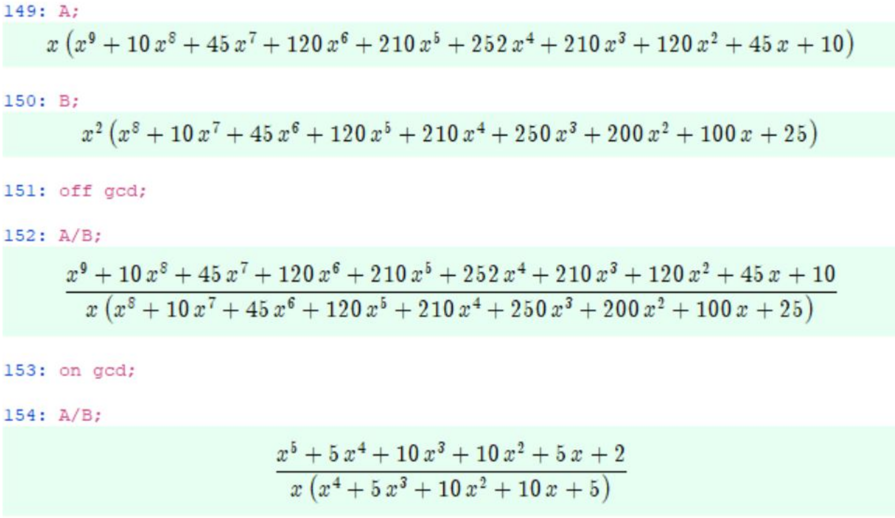
James Davenport

University of Bath

5 September 2018

Why GCD

Because otherwise every calculations with fractions tends to blow up.



```
149: A;
```

$$x\left(x^9+10\,x^8+45\,x^7+120\,x^6+210\,x^5+252\,x^4+210\,x^3+120\,x^2+45\,x+10\right)$$

```
150: B;
```

$$x^2\left(x^8+10\,x^7+45\,x^6+120\,x^5+210\,x^4+250\,x^3+200\,x^2+100\,x+25\right)$$

```
151: off gcd;

152: A/B;
```

$$\frac{x^9+10\,x^8+45\,x^7+120\,x^6+210\,x^5+252\,x^4+210\,x^3+120\,x^2+45\,x+10}{x\left(x^8+10\,x^7+45\,x^6+120\,x^5+210\,x^4+250\,x^3+200\,x^2+100\,x+25\right)}$$

```
153: on gcd;

154: A/B;
```

$$\frac{x^5+5\,x^4+10\,x^3+10\,x^2+5\,x+2}{x\left(x^4+5\,x^3+10\,x^2+10\,x+5\right)}$$

Modular Algorithms: Rationale

On our example, even the Primitive algorithm, which removes all the content, still has to compute $-845588889704275438942 99$ as the last remainder, before deciding that the result does not depend on x, so any g.c.d. is merely explained by the contents.
Can we do better?

Modular Algorithms: An example

Suppose that these two polynomials have a common factor, that is a polynomial P (of non-zero degree) which divides A and B. Then there is a polynomial Q such that $A = PQ$. This equation still holds if we take each coefficient as an integer modulo 5.
If we write P_5 to signify the polynomial P considered as a polynomial with coefficients modulo 5, this equation 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 $P = 1$.

Questions

How do we turn this into an algorithm?

1. how do we calculate a non-trivial g.c.d.?
2. what do we do if the modular g.c.d. is not the modular image of the g.c.d.

N.B. This happens if we take p=2: both A and B have $x + 1$ as a factor $\pmod 2$

3. how much does this method cost?

It turns out that, to answer any of these, we need to know one thing.

How big can the coefficients be?

It may seem strange, but the coefficients of a g.c.d. of two polynomials can be greater than the coefficients of the polynomials themselves. One example which shows this is the following (due to Davenport and Trager):

$$\begin{aligned}
A = &\quad x^3 + x^2 - x - 1 &&= (x+1)^2(x-1);\\
B = &\quad x^4 + x^3 + x + 1 &&= (x+1)^2(x^2-x+1);\\
\gcd(A,B) = &\quad x^2 + 2x + 1 &&= (x+1)^2.
\end{aligned}$$

This example can be generalised, as say

$$\begin{aligned}
A = &\quad x^5 + 3x^4 + 2x^3 - 2x^2 - 3x - 1 &&= (x+1)^4(x-1);\\
B = &\quad x^6 + 3x^5 + 3x^4 + 2x^3 + 3x^2 + 3x + 1 &&= (x+1)^4(x^2-x+1)\\
\gcd(A,B) = &\quad x^4 + 4x^3 + 6x^2 + 4x + 1 &&= (x+1)^4.
\end{aligned}$$

Landau–Mignotte Inequality

Theorem (Landau–Mignotte Inequality [Lan05, Mig74, Mig82])

Let $Q = \sum_{i=0}^{q} b_i x^i$ be a divisor of the polynomial $P = \sum_{i=0}^{p} a_i x^i$ (where a_i and b_i are integers). Then

$$\max_{i=0}^{q} |b_i| \le \sum_{i=0}^{q} |b_i| \le 2^q \left|\frac{b_q}{a_p}\right| \sqrt{\sum_{i=0}^{p} a_i^2} = 2^q \left|\frac{b_q}{a_p}\right| ||A||.$$

When it comes to g.c.d.s, we have the following result.

Corollary

Every coefficient of the g.c.d. of $A = \sum_{i=0}^{\alpha} a_i x^i$ and $B = \sum_{i=0}^{\beta} b_i x^i$ (with a_i and b_i integers) is bounded by

$$2^{\min(\alpha,\beta)} \gcd(a_\alpha, b_\beta) \min\left(\frac{1}{|a_\alpha|}||A||, \frac{1}{|b_\beta|}||B||\right).$$

The modular – integer relationship (1)

Lemma

If p does not divide the leading coefficient of $\gcd(A, B)$, the degree of $\gcd(A_p, B_p)$ is greater than or equal to that of $\gcd(A, B)$.

This is not much use, because we are trying to *find* the g.c.d.

Corollary (LC Corollary)

If p does not divide the leading coefficients of A and of B (it may divide one, but not both), then the degree of $\gcd(A_p, B_p)$ is greater than or equal to that of $\gcd(A, B)$.

As the g.c.d. is the only polynomial (to within an integer multiple) of its degree which divides A and B, we can test the correctness of our calculations of the g.c.d.: if the result has the degree of $\gcd(A_p, B_p)$ (where p satisfies the hypothesis of this corollary) and if it divides A and B, then it is the g.c.d. (to within an integer multiple).

Sylvester Matrix (as in "MatrixQuestion")

We write $f = \sum_{i=0}^{n} a_i x^i$ and $g = \sum_{i=0}^{m} b_i x^i$.

Definition

The *Sylvester matrix* of f and g is the matrix $\mathrm{Syl}(f, g) =$

$$\begin{pmatrix}
a_n & a_{n-1} & \dots & a_1 & a_0 & 0 & 0 & \dots & 0 \\
0 & a_n & a_{n-1} & \dots & a_1 & a_0 & 0 & \dots & 0 \\
\vdots & \ddots & \ddots & \ddots & \dots & \ddots & \ddots & \ddots & \vdots \\
0 & \dots & 0 & a_n & a_{n-1} & \dots & a_1 & a_0 & 0 \\
0 & \dots & 0 & 0 & a_n & a_{n-1} & \dots & a_1 & a_0 \\
b_m & b_{m-1} & \dots & b_1 & b_0 & 0 & 0 & \dots & 0 \\
0 & b_m & b_{m-1} & \dots & b_1 & b_0 & 0 & \dots & 0 \\
\vdots & \ddots & \ddots & \ddots & \dots & \ddots & \ddots & \ddots & \vdots \\
0 & \dots & 0 & 0 & b_m & b_{m-1} & \dots & b_1 & b_0
\end{pmatrix}$$

where there are m lines constructed with the a_i, n lines constructed with the b_i.

Sylvester Matrix continued)

Then Gaussian elimination in the Sylvester matrix corresponds to subtracting (scaled and shifted) copies of one polynomial from the other.

Definition

The *resultant* of f and g, written $\mathrm{Res}(f, g)$ (or $\mathrm{Res}_x(f, g)$) is the determinant of the Sylvester matrix.

Proposition

$\mathrm{Res}(f, g) = 0 \Leftrightarrow f$ *and* g *have a common factor.*

The modular – integer relationship (2)

Lemma

Let $C = \gcd(A, B)$. If p satisfies the condition of the LC corollary, and if p does not divide $\mathrm{Res}_x(A/C, B/C)$, then $\gcd(A_p, B_p) = C_p$.

We don't know the g.c.d., so we don't know $\mathrm{Res}_x(A/C, B/C)$, but it's some non-zero number, so has only finitely primes that divide it.

Definition

If $\gcd(A_p, B_p) = \gcd(A, B)_p$, we say that the reduction of this problem modulo p is *good*, or that p is *of good reduction*. If not, we say that p is *of bad reduction*.

The modular – integer relationship (3)

Theorem (Good Reduction Theorem ($\mathbf{Z}$))

If p does not divide $\gcd(a_\alpha, b_\beta)$ (which can be checked for in advance) or $\mathrm{Res}_x(A/C, B/C)$, then p is of good reduction. Furthermore, if p divides $\mathrm{Res}_x(A/C, B/C)$ but not $\gcd(a_\alpha, b_\beta)$, then the gcd computed modulo p has a larger *degree than the true result.*

So (after making this check!), we can only be wrong in one direction!
It two primes disagree, the one with the larger degree is definitely wrong!

Modular GCD (Large prime version)

Algorithm (Modular GCD (Large prime version))

```
1: procedure LPGCD($A, B$)                                ▷ $A, B$ monic in $\mathbf{Z}[x]$.
2:     $M$ :=Landau_Mignotte_bound($A, B$);
3:     while true do $p$ :=find_large_prime($2M$);
4:         if $p$ does not divide gcd($\mathrm{lc}(A), \mathrm{lc}(B)$) then
5:             $C$ :=modular_gcd($A, B, p$);
6:             if $C$ divides $A$ and $C$ divides $B$ then
7:                 Return $C$
8:             end if
9:         end if
10:    end while                           ▷ Theorem guarantees termination
11: end procedure
```

Modular GCD (Large prime version)

Figure: Diagrammatic illustration of Algorithm 1

$$\begin{tikzcd}
\mathbf{Z}[x] \arrow[r, dashed] \arrow[d, "\text{reduce}"'] & \mathbf{Z}[x] \\
\mathbf{Z}_p[x] \arrow[r, "\text{gcd}"] & \mathbf{Z}_p[x] \arrow[u, "\text{interpret \& check}"']
\end{tikzcd}$$

`modular_gcd` is a “Monte Carlo” algorithm (sometimes right, sometimes wrong), and this makes it “Las Vegas” (always right, probably fast).

Modular GCD (non-monic polynomials)

If the inputs are not monic, the answer might not be monic. For example, $x+4$ divides both $2x^2+x$ and $2x^2-x-1$ modulo 7, but the true common divisor over the integers is $2x+1$, which is $2(x+4) \pmod 7$.
We do know that the leading coefficient of the g.c.d. divides each leading coefficient $\mathrm{lc}(A)$ and $\mathrm{lc}(B)$, and therefore their g.c.d. $g = \gcd(\mathrm{lc}(A), \mathrm{lc}(B))$. We therefore compute

$$C := \mathrm{pp}(\underbrace{g \times \texttt{modular\_gcd}(A,B)}_{\substack{\text{computed modulo } M,\\ \text{then interpreted in } \mathbf{Z}}}) \tag{1}$$

instead, where the pp is needed in case the leading coefficient of the g.c.d. is a proper factor of g.

How costly?

It is tempting to argue that this algorithm will only handle numbers of the size of twice the Landau–Mignotte bound, but this belief has two flaws.

- While we have proved that there are only finitely many bad primes, we have said nothing about how many there are. The arguments can in fact be made effective, but the results tend to be unduly pessimistic, since it is extremely unlikely that all the bad primes would be clustered just above $2M$.
- In theory, and indeed very often in practice [ABD88], the division tests could yield very large numbers if done as tests of the remainder being zero: for example the remainder on dividing x^{100} by $x - 10$ is 10^{100}. This can be solved by a technique known as "early abort" trial division.

Early abort

Proposition (Based on Landau–Mignotte)

If h, of degree m, is a factor of f of degree n, the coefficient of x^{n-m-i} in the quotient is bounded by $\binom{n-m}{i}\frac{1}{\mathrm{lc}(h)}||f||$.

Hence, as we are doing the trial division, we can give up as soon as we find a coefficient in the quotient that exceeds this bound.
For example, if we take $p = 7$ in our example, we find that the g.c.d. of A_7 and B_7 is $x + 10$. Does $x + 10$ divide B?
We note that $||B|| \approx 23.92$.
Successive terms in the quotient are $3x^5$ (and 3 is a permissible coefficient), $-30x^4$ (and $30 < \binom{5}{1} \times 23.92$) and $305x^3$.
We observe that $305 > \binom{5}{2} \times 23.92 = 239.2$, so this *cannot* be a divisor of B. Hence 7 was definitely unlucky.
With this refinement, it is possible to state that the numbers dealt with in this algorithm are "not much larger" than $2M$.

Example (same as before)

$\sqrt{\sum_{i=0}^{8} a_i^2} = \sqrt{113}$, $\sqrt{\sum_{i=0}^{6} b_i^2} = 2\sqrt{143}$, and hence corollary 2 gives a bound of

$$2^6 \min\left(\sqrt{113}, \frac{2}{3}\sqrt{143}\right) \approx 510.2, \tag{2}$$

so our first prime would be 1021, which is indeed of good reduction.
If we turn the polynomials round
$x^8+x^6-3x^4-3x^3+8x^2+2x-5 \mapsto 1+x^2-3x^4-3x^5+8x^6+2x^7-5x^8$
etc., then

$$2^6 \min\left(\frac{1}{5}\sqrt{113}, \frac{2}{21}\sqrt{143}\right) \approx 72.8, \tag{3}$$

so our first prime would be 149.
In general we compute both bounds! But 5 was definitely too small!

Explanation on Turning Round

$A(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$: let
$\hat{A}(x) = x^8 A(1/x) = 1 + x^2 - 3x^4 - 3x^5 + 8x^6 + 2x^7 - 5x^8$.
Suppose $G(x) = \gcd(A(x), B(x))$, so $A = GC$ and $B = GD$ for some polynomials C, D. Then (as it's the same arithmetic on coefficients), $\hat{A} = \hat{G}\hat{C}$ and $\hat{B} = \hat{G}\hat{D}$. So $\hat{G}$ is a common divisor of $\hat{A}$ and $\hat{B}$.
In fact it's the *greatest*, because if a larger $\hat{H}$ really is the g.c.d. of $\hat{A}$ and $\hat{B}$, then H has to divide A and B: so G isn't their g.c.d.
So in terms of finding the answer, we can work with either A, B or $\hat{A}, \hat{B}$.

Chinese Remainder Theorem 孙子算经

"There are certain things whose number is unknown. If we count them by threes, we have two left over; by fives, we have three left over; and by sevens, two are left over. How many things are there?"

Theorem (Chinese Remainder Theorem (coprime form))

Two congruences

$$X \equiv a \pmod{M} \tag{4}$$

$$X \equiv b \pmod{N}, \tag{5}$$

where M and N are relatively prime, are precisely equivalent to one congruence

$$X \equiv c \pmod{MN}. \tag{6}$$

By this we mean that, given any a, b, M and N, we can find such a c that satisfaction of (4) and (5) is precisely equivalent to satisfying (6).

Algorithm (Modular GCD (Small prime version))

```
1: procedure $\mathrm{SPGCD}(A, B)$                ▷ $A, B$ monic in $\mathbf{Z}[x]$.
2:     $M :=$LMbound$(A, B)$; $g := \gcd(\mathrm{lc}(A), \mathrm{lc}(B))$;
3:     $N := p :=$ find_prime$(g)$; $D :=$ gmodular_gcd$(A, B, p)$;
4:     if $\deg(D) = 0$ then return 1
5:     end if
6:     while $N < 2M$ do
7:         $p :=$ find_prime$(g)$; $C :=$ gmodular_gcd$(A, B, p)$;
8:         if $\deg(C) = \deg(D)$ then
9:             $D := CRT(C, D, p, N)$; $N := pN$;
10:        else if $\deg(C) < \deg(D)$ then        ▷ $C$ proves $D$ is bad
11:            if $\deg(C) = 0$ then Return 1
12:            end if
13:            $D := C$; $N := p$;
14:        else                ▷ $D$ proves that $p$ is bad, so we ignore it
15:        end if
16:    end while                        ▷ Theorem guarantees termination
17:    $D := \mathrm{pp}(D)$            ▷ In case multiplying by $g$ was overkill
```

Modular GCD (Small prime version) continued

We still need to check that D divides A and B, and return it.If not, all primes must have been bad, and we start again.
We still need "early abort" division checks.
We should note the heavy reliance on Corollary 4 to detect bad reduction.

Diagrammatic illustration

Figure: Diagrammatic illustration of Algorithm 2

$$\begin{tikzcd}
\mathbf{Z}[x] \arrow[rrrr, dashed] \arrow[d, "k\times\text{reduce}"'] & & & & \mathbf{Z}[x] \\
\mathbf{Z}_{p_1}[x] \arrow[r, "\text{gcd}"] & \mathbf{Z}_{p_1}[x] & & & \\
\vdots & \vdots & \left.\phantom{\begin{matrix}a\\a\\a\end{matrix}}\right\} \arrow[r, "\text{C.R.T.}"] & & \mathbf{Z}'_{p_1\cdots p_k}[x] \arrow[uu, "\text{interpret \& check}"'] \\
\mathbf{Z}_{p_k}[x] \arrow[r, "\text{gcd}"] & \mathbf{Z}_{p_k}[x] & & &
\end{tikzcd}$$

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

Various improvements

Many improvements are being made to this algorithm.
The LMBound, while true, is often much too big. *Any* G that divides A and B, and has the degree of a modular g.c.d. that satisfies the LC Corollary, is actually a *greatest* common divisor.
So replace line 9 "$D := \mathrm{CRT}(C, D, p, N)$; $N := pN$;" by

Algorithm (Line 9 replacement)

```
$D_1 := CRT(C, D, p, N)$; $N := pN$;
if $D = D_1$ & $D$ divides $A$ and $D$ divides $B$ then
    return $D$                    ▷ A GREATEST common divisor
else
    $D := D_1$
end if
```

Choices?

A choice between one large prime (Algorithm 1) and several small ones (Algorithms 2)?
In practice, of course, a computer regards all numbers less than 32 bits (and increasingly 64 bits) as ‘small’, so an implementation would generally use the largest ‘small’ primes it could in Algorithm 2, and thus often one prime will suffice, especially if we have an “early termination” test replacing lines 4,5.

Beyond $\mathbf{Z}[x]$: the problem (1)

Consider $\mathbf{Z}[y][x]$ and

$$\begin{aligned} A &= (y^2 - y - 1)x^2 - (y^2 - 2)x + (2y^2 + y + 1); \\ B &= (y^2 - y + 1)x^2 - (y^2 + 2)x + (y^2 + y + 2). \end{aligned}$$

The first elimination gives

$$C = (2y^2 - 4y)x + (y^4 - y^3 + 2y^2 + 3y + 3),$$

and the second gives

$$\begin{aligned} D = & -y^{10} + 3\,y^9 - 10\,y^8 + 11\,y^7 - 23\,y^6 + 22\,y^5 - 37\,y^4 \\ & +29\,y^3 - 32\,y^2 + 15\,y - 9. \end{aligned}$$

Since this is a polynomial in y only, the greatest common divisor does not depend on x, i.e. $\mathrm{pp}_x(\gcd(A, B)) = 1$. Since $\mathrm{cont}_x(A) = 1$, $\mathrm{cont}_x(\gcd(A, B)) = 1$, and hence $\gcd(A, B) = 1$, *but* we had to go via a polynomial of degree 10 to show this.

Beyond $\mathbf{Z}[x]$: the problem (2)

Even with subresultant methods, if A and B have degree d in x and y, intermediate results can have degree d^2.
Now suppose our polynomials were in $n+1$ variables x and $y_1, \ldots, y_n$. Then the coefficients (with respect to x) of the initial polynomials would have degree d_y, and hence (at most) $(1+d_y)^n$ terms. But the result would have $(1+2d_xd_y)^n$ terms, roughly $(2d_x)^n$, or exponentially many, times as many terms as the inputs.
My experience suggests that, even if f and g are sparse, i.e. have many fewer than $(1+d_y)^n$ terms, the intermediate results are dense, and so the blowup *ratio* is even worse.

Beyond $\mathbf{Z}[x]$: modular methods?

We consider the bivariate case, $\gcd(A, B)$ with $A, B \in R[x, y] \equiv R[y][x]$, and we will be considering evaluation maps replacing y by $v \in R$, writing A_{y-v} for the result of this evaluation.

Lemma

If $y - v$ does not divide the leading coefficient (in x) of $\gcd(A, B)$, the degree (in x) of $\gcd(A_{y-v}, B_{y-v})$ is greater than or equal to that of $\gcd(A, B)$.

Corollary

If $y - v$ does not divide the leading coefficients of A and of B (it may divide one, but not both), then the degree (in x) of $\gcd(A_{y-v}, B_{y-v})$ is greater than or equal to that of $\gcd(A, B)$.

Beyond $\mathbf{Z}[x]$: modular methods (2)

As the g.c.d. is the only polynomial (to within a multiple from $R[y]$) of its degree (in x) which divides A and B, we can test the correctness of our calculations of the g.c.d.: if the result has the degree of $\gcd_x(A_{y-v}, B_{y-v})$ (where v satisfies the hypothesis of this corollary) and if it divides A and B, then it is the g.c.d. (to within a multiple from $R[y]$).

Lemma

Let $C = \gcd(A, B)$. If v satisfies the condition of the corollary, and if $y - v$ does not divide $\mathrm{Res}_x(A/C, B/C)$, then $\gcd(A_{y-v}, B_{y-v}) = C_{y-v}$.

Definition

If $\gcd(A_{y-v}, B_{y-v}) = \gcd(A, B)_{y-v}$, we say that the evaluation of this problem at v is *good*, or that $y - v$ is *of good reduction*. If not, we say that $y - v$ is *of bad reduction*.

Algorithm

Algorithm 21 (Bivariate Modular GCD)
Input: *A, B primitive polynomials in $\mathbf{Z}_p[y[[x]$.*
Output: $\gcd(A, B)$

```
$g := \gcd(\mathrm{lc}_x(A), \mathrm{lc}_x(B))$;
$v :=$ find_value($g$);
$D :=$ gmodular_gcd($A_{y-v}, B_{y-v}, p$);
if $\deg(D) = 0$ then return 1
$N := y - v$;        # $N$ is the modulus we will be constructing
while $\deg_y(N) \le \min(\deg_y(A), \deg_y(B))$ repeat
        $v :=$ find_value($g$);
        $C :=$ gmodular_gcd($A_{y-v}, B_{y-v}, p$);
        if $\deg(C) = \deg(D)$
           then $D :=$ Algorithm 50($C, D, y - v, N$);
                  $N := (y - v)N$;
           else if $\deg(C) < \deg(D)$
                    # $C$ proves that $D$ is based on values of bad reduction
                    if $\deg(C) = 0$ then return 1
                    $D := C$;
                    $N := y - v$;
           else     #$D$ proves that $v$ is of bad reduction, so we ignore it
$D := \mathrm{pp}(D)$;     # In case multiplying by $g$ was overkill
Check that $D$ divides $A$ and $B$, and return it
If not, all values must have been bad, and we start again
```

find_value(g) finds a new value v each time, such that g_{y-v} does not vanish. It is conceivable that we will exhaust $\mathbf{Z}_p$, in which case we have to move to choosing v from an algebraic extension of $\mathbf{Z}_p$. Indeed it is possible that there are no values v in $\mathbf{Z}_p$ such that g_{y-v} does not vanish, e.g. $g = y(y-1)(y-2)$ when $p = 3$.

Diagram

Figure: Diagrammatic illustration of Algorithm

$$\begin{tikzcd}
\mathbf{Z}_p[y][x] \arrow[rrr, dashed] \arrow[d, "k\times\text{reduce}"'] & & & \mathbf{Z}_p[y][x] \\
\mathbf{Z}_p[y]_{y-v_1}[x] \arrow[r, "\text{gcd}"] & \mathbf{Z}_p[y]_{y-v_1}[x] & & \\
\vdots & \vdots & \vdots \arrow[r, "\text{C.R.T.}"] & \dfrac{\mathbf{Z}_p[y][x]}{\prod'(y-v_1)\cdots(y-v_k)} \arrow[uu, "\text{interpret}"', "\text{\& check}"] \\
\mathbf{Z}_p[y]_{y-v_k}[x] \arrow[r, "\text{gcd}"] & \mathbf{Z}_p[y]_{y-v_k}[x] & &
\end{tikzcd}$$

$\prod'$ indicates that some of the v_i may have been rejected by the compatibility checks, so the product is over a subset of $(y-v_1)\cdots(y-v_k)$.

General Complexity Result

Theorem ([Bro71, (95)])

Suppose $f, g \in \mathbf{Z}[x_1, \ldots, x_n]$. Let l bound the lengths of the integer coefficients of f and g, and d bound their degree in each of the x_i. Then under the following assumptions:

1. *classical $O(N^2)$ multiplication and division;*
2. *no bad values are found, either for primes or for evaluations;*
3. *that the answer/quotients have coefficient lengths at most l;*
4. *that we can use single-word primes;*

the running time of the 'modular first'algorithm is

$$O\left(l^2(d+1)^n + (d+1)^{n+1}\right).$$

This contrasts with the subresultant algorithm's bound

$$O\left(l^2(d+1)^{4n}2^{2n^2}3^n\right),$$

General Complexity Result (Comments

The dependence on $(d+1)^n$ has essentially gone from quartic to linear. $(d+1)^n$ is the maximal number of terms in the input, so this "modular first" algorithm is "almost optimal" for dense inputs.

Bibliography I

J.A. Abbott, R.J. Bradford, and J.H. Davenport.
Factorisation of Polynomials: Old Ideas and Recent Results.
In R. Janssen, editor, *Proceedings “Trends in Computer Algebra”*, pages 81–91, 1988.

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

E. Landau.
Sur Quelques Théorèmes de M. Petrovic Relatif aux Zéros des Fonctions Analytiques.
Bull. Soc. Math. France, 33:251–261, 1905.

Bibliography II

M. Mignotte.
An Inequality about Factors of Polynomials.
Math. Comp., 28:1153–1157, 1974.

M. Mignotte.
Some Useful Bounds.
Symbolic and Algebraic Computation (Computing Supplementum 4) Springer-Verlag, pages 259–263, 1982.