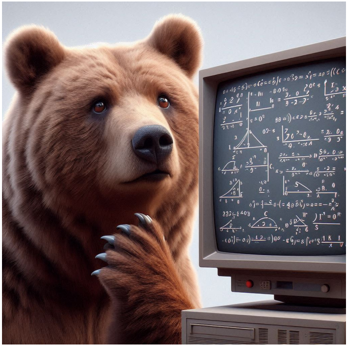
What Even is a Computer Algebra?

Mitchell Holt

August 2024



Exact Computation

Example

$$\underbrace{\frac{1}{10} + \frac{1}{10} + \cdots + \frac{1}{10}}_{10 \text{ times}}$$

Python:

```
>>> total = 0
>>> for i in range(10):
...     total += 1/10
...
>>> total
0.9999999999999999
```

Exact Computation

Example

$$\underbrace{\frac{1}{10} + \frac{1}{10} + \cdots + \frac{1}{10}}_{\text{10 times}}$$

Maple:

```
> total := 0:
> for i to 10 do
>     total += 1/10;
> end do:
> total;
1
```

Representing Data in Computer Algebra

(i) Use arbitrary precision integers

(ii) Now we can easily represent fractions - use GCD to keep the fraction in lowest terms and avoid *expression swell*.

(iii) DO NOT USE FLOATS I SWEAR I WILL KNOW AND I WILL FIND YOU

Exact Computation

Example

$$(\sqrt{2})^2$$

Python:

```
>>> math.sqrt(2) ** 2
2.0000000000000004
```

Maple:

```
> sqrt(2)^2
2
```

Representing Data in Computer Algebra

(i) An *algebraic* number is the root of a polynomial with rational coefficients. For example, $\sqrt{2}$ is a root of $x^2 - 2$.

(ii) We can represent irrational algebraic numbers with their *minimal polynomial*, which is just the (unique) smallest polynomial it is a root of.

Polynomials

Definition (Polynomials over a Field)

Let F be a *field*. The polynomials over F in the symbol x (write $F[x]$) are sums

$$a = \sum_{i=0}^{m} a_i x^i, \quad a_0, \ldots, a_m \in F,\ a_m \neq 0$$

Here, m is the *degree* of a.

Our Goal: Polynomial GCD

Definition (Polynomial GCD)

Let F be a field and $a, b \in F[x]$ be polynomials that are not both zero. The *greatest common divisor* of a and b (write $\gcd(a, b)$) is the *monic* is the maximum polynomial (w.r.t. degree) dividing both a and b.

“Why Should I Care?”

Chat GPT says I should mention:

(i) Applications in computer algebra

(ii) Algebraic geometry

(iii) Public key cryptography

(iv) Educational value

Polynomial Representations

Dense Representation

(i) Store a list (or vector or array) of coefficients,e.g. $x^3 - x + 2$ becomes `[2, -1, 0, 1]`.

(ii) Store the degree of the polynomial.

(iii) Efficient access to *every* coefficient.

(iv) What about the zero polynomial?

Sparse Representation

(i) Store a list of *monomials*

(ii) Efficient when polynomials are... sparse, e.g. $x^{10000} - 1$.

Algorithm: Polynomial Addition

Input: *Dense* polynomials $a, b \in F[x]$
Output: $c = a + b \in F[x]$ and $\deg c$

```
if a = 0 (b = 0) then
    return b (return a)
end if
deg c ← −1
N ← max(deg a, deg b)
for i = 0, 1, ..., N do
    c_i ← a_i + b_i in F
    if c_i ≠ 0 then
        deg c ← i
    end if
end for
return [c_0, c_1, ..., c_N] and deg c
```

Complexity *in field operations*? $O(N)$.

Polynomial Multiplication

Example

$(x^2 + 5x + 1)(x - 2) \in \mathbb{Q}[x]$

(i) What is the degree of the product?

(ii) What is the coefficient of x in the product?

Let

$$a = \sum_{i=0}^{m} a_i x^i \quad \text{and} \quad b = \sum_{j=0}^{n} b_j x^j$$

Then

$$ab = \sum_{k=1}^{m+n} c_k x^k \quad \text{where } c_k = \sum_{r=0}^{k} a_r b_{k-r}$$

Algorithm: Polynomial Multiplication

Input: *Dense* polynomials $a, b \in F[x]$
Output: $c = ab \in F[x]$

$N \leftarrow \deg a + \deg b$
for $k = 0, 1, \ldots, N$ **do**
  $c_k \leftarrow 0$
  for $r = 0, 1, \ldots, k$ **do**
    $c_k \leftarrow c_k + a_r b_{k-r}$
  end for
end for
return $[\, c_0, c_1, \ldots, c_N]$

Algorithm Analysis

Recall

$$ab = \sum_{k=0}^{m+n} c_k x^k, \quad \text{where} \quad c_k = \sum_{r=0}^{k} a_r b_{k-r}$$

(i) c_k requires $k+1$ field multiplications.

(ii) Hence ab requires

$$\sum_{k=0}^{m+n} (k+1) = \frac{(m+n+1)(m+n+2)}{2}$$

field multiplications.

If $N = \deg ab = m + n$, then multiplication is $O(N^2)$.

Polynomial Division

Theorem (Euclidean Divison)

Let $a, b \in F[x]$. Then there exists $q, r \in F[x]$ with $a = bq + r$, with $r = 0$ or $\deg r < \deg b$.

Example

Let $a, b \in \mathbb{Z}_7[x]$ be given by

$$a = 5x^5 + 4x^4 + 3x^3 + 2x^2 + x,$$
$$b = x^2 + 2x + 3$$

Then $q = 5x^3 + x^2 + 6$ and $r = 3x + 3$.

Synthetic Division (Polynomial Long Division)

No way I'm doing this in LaTeX

Fast Division

Let $m = \deg a$, $n = \deg b$, and $a = bq + r$. *Remember*:

(i) $\deg q = m - n$

(ii) $r = 0$ or $\deg r < n$

Algorithm Idea:

1. Compute $b^{-1} \mod x^{m-n+1}$.
2. Recover q.
3. Compute $r = a - bq$.

$$a = bq + r \mod x^{m-n+1}$$
$$q = b^{-1}(a - r) \mod x^{m-n+1}$$

Problem: $r \mod x^{m-n+1}$ might not be 0, so we need to know r *a priori*.

Fast Division

Idea: *Reverse* the polynomials!

$$\mathrm{rev}_m(a(x)) = x^m a\left(\frac{1}{x}\right) = \sum_{k=0}^{m} a_{m-k} x^k$$

Let $a = bq + r$, with $\deg a = m$ and $\deg q = n$.

$$\begin{aligned} x^m a\left(\frac{1}{x}\right) &= x^m b\left(\frac{1}{x}\right) q\left(\frac{1}{x}\right) + x^m r\left(\frac{1}{x}\right) \\ &= x^n b\left(\frac{1}{x}\right) \cdot x^{m-n} q\left(\frac{1}{x}\right) + x^{m-n+1} \cdot x^{n-1} r\left(\frac{1}{x}\right) \\ \mathrm{rev}_m(a) &= \mathrm{rev}_n(b)\,\mathrm{rev}_{m-n}(q) + x^{m-n+1}\,\mathrm{rev}_{n-1}(r) \end{aligned}$$

Then

$$\mathrm{rev}_m(a) \equiv \mathrm{rev}_n(b)\,\mathrm{rev}_{m-n}(q) \mod x^{m-n+1}$$

Fast Division

$$\mathrm{rev}_m(a) \equiv \mathrm{rev}_n(b)\,\mathrm{rev}_{m-n}(q) \mod x^{m-n+1}$$

$$\mathrm{rev}_{m-n}(q) \equiv \mathrm{rev}_n(b)^{-1}\,\mathrm{rev}_m(a) \mod x^{m-n+1}$$

(i) $\deg(\mathrm{rev}_{m-n}(q)) = \deg q = m - n$, so we have $\mathrm{rev}_{m-n}(q)$ proper.

(ii) $\mathrm{rev}_{m-n}(\mathrm{rev}_{m-n}(q)) = q$.

Fast Division

Input: polynomials $a, b \in F[x]$
Output: q, r with $a = qb + r$ such that $r = 0$ or $\deg r < \deg b$

$m,\ n \leftarrow \deg a,\ \deg b$
$a_r,\ b_r \leftarrow \mathrm{rev}_m(a),\ \mathrm{rev}_n(b)$
$b'_r \leftarrow b_r^{-1} \mod x^{m+n-1}$
$q \leftarrow \mathrm{rev}_{m-n}(b'_r a_r \mod x^{m+n-1})$
$r \leftarrow a - bq$
return $q,\ r$

Truncated Power Series Inverse

Let $f \in F[x]$ and $g \in F[x]$ *approximate* f^{-1} with $fg \equiv 1 \mod x^k$ Observe:

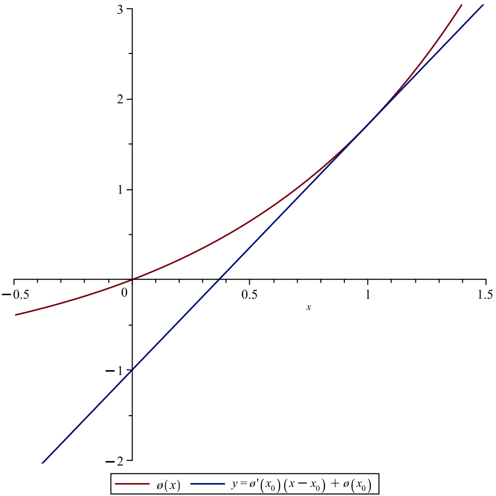
(i) $g(0) = f(0)^{-1}$

(ii) $g = g(0)$ is an inverse $\mod x^1$

Question: From this initial approximation, can we *successively* get better approximations?

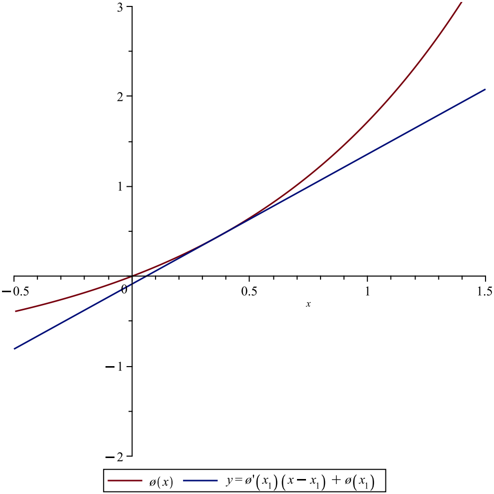
Newton Iteration

Initial guess: $x_0 = 1$



Newton Iteration

Next guess: $x_1 = x_0 - \dfrac{\phi(x_0)}{\phi'(x_0)} \approx 0.3678794412$



Truncated Power Series Inverse

Let

$$\phi(g) = 1/g - f$$

Then

$$g_{i+1} = g_i - \frac{1/g_i - f}{-1/g_i^2} = 2g_i - fg_i^2$$

Theorem

Let $f, g_0, g_1, \dots \in F[x]$ with $g_0 = f(0)^{-1}$ and $g_{i+1} = 2g_i - fg_i^2 \mod x^{2^{i+1}}$ for all i. Then $fg_i \equiv 1 \mod x^{2^i}$ for all $i \geq 0$.

Proof.

Induction on i. □

Fast Newton Iteration

Input: $f \in F[x]$ and $k \in \mathbb{N}$
Output: $g \in F[x]$ with $fg \equiv 1 \mod x^k$

$g_0 \leftarrow f(0)^{-1}$
$r \leftarrow \lceil \log_2 k \rceil$
for $i = 1, \ldots, r$ **do**
  $g_i \leftarrow 2g_{i-1} - fg_{i-1}^2 \mod x^{2^i}$
end for
return g_r

Proposition (Complexity of Fast Division)

Let $M(n)$ be the number of multiplications in F needed to multiply two polynomials of degree $n-1$ in $F[x]$. Let $D(n)$ be the number of multiplications in F to compute $f^{-1} \mod x^n$. If M is *super-linear*, $D(n) < 3M(n) + O(n)$.

Proof.

lmk if u care and i'll send one $\square$

Fast Division

Corollary

The fast division algorithm costs around the same as 4 polynomial multiplications.

Polynomial GCD

Lemma (probably Euclid)

Let $a, b, q, r \in F[x]$ and suppose that we have the Euclidean divisions

$$a = bq + r$$

Then $\gcd(a, b) = \gcd(b, r)$.

Polynomial GCD

Example

$$\begin{aligned}
\gcd(x^3+1,\, x^2-1) &\qquad x^3+1 = (x^2-1)x + (x+1) \\
\gcd(x^2-1,\, x+1) &\qquad x^2-1 = (x+1)(x-1) + 0 \\
\gcd(x+1,\, 0) &\qquad x+1
\end{aligned}$$

Polynomial GCD

Input: $a, b \in F[x]$, $a \neq 0$, with $b = 0$ or $\deg a > \deg b$
Output: $\gcd(a, b)$

```
if b = 0 then
    return a
end if
q, r ← Euclidean division of a by b
return gcd(b, r)
```

Notice that $r = 0$ or $\deg r < \deg b$, proving termination.

Thanks for Listening!