Detecting Polynomial Perfect Powers

1 Lacunary Polynomials

We seek algorithms that are fast when the input is given in the lacunary representation.

For \( f \in \mathbb{R}[x_1, \ldots, x_n] \) with total degree \( n \), \n given by

\[
f(x) = c \alpha_0^nx_0 + c \alpha_1^nx_1 + \cdots + c \alpha_n^nx_n,
\]

where each \( c_i \in \mathbb{R} \setminus \{0\} \) and \( x^k = x_1^{k_1} x_1^{k_2} \cdots x_n^{k_n} \), we store only a linked list of tuples \((c_i, e_i)\), for a total size of \( O(\log n) \).

So by ‘fast’, we mean complexity polynomial in \( t \log n \), and in the case that \( R = \mathbb{Q} \), polynomial in the height of \( f \) as well.

2 Are You Perfect?

Question. Given a multivariate polynomial \( f \in \mathbb{R}[x_1, \ldots, x_k] \), how do we determine if \( f \) is a perfect power?

That is, are there \( h \in \mathbb{R}[x_1, \ldots, x_k] \) and \( r \geq 2 \) such that \( f = h^r \)?

3 Summary of Results

For multivariate polynomials over \( \mathbb{Q} \) or a finite field with sufficiently large characteristic, we present Monte Carlo algorithms to detect lacunary polynomials which are perfect powers.

That is, our algorithms are correct with controllably high probability and always fast.

4 Integer Polynomials Algorithm

Input: \( f \in \mathbb{Z}[x] \)

Output: An \( r \) such that \( f \) is an \( r \)-th power, or ‘FALSE’ if \( f \) is not a perfect power.

1. For each possible prime power \( r \) do
   2. Pick a prime \( p \) with \( f \mod p \)
   3. Pick a prime power \( q \) with \( q = 1 \)
   4. Choose random \( a_1, \ldots, a_n \in \mathbb{Z} \)
   5. If \( f(a_1, \ldots, a_n) = \cdots = f(a_1, \ldots, a_n) = 1 \) over \( \mathbb{F} \)
   6. return \( r \)
   7. end do
   8. return ‘FALSE’

Boring Details

- Steps 2-6 will actually be repeated \( O(\log 1/r) \) times to guarantee success with probability \( 1 - e \).
- For Step 3, we can either find a random prime \( q \) such that \( p \mid q \) and \( r \mid (q - 1) \), or choose \( q = p - 1 \).
- Use an extension field. The first approach yields better practical performance than polynomial theoretical results.

- If \( f \in \mathbb{F} \), for some prime power \( p \), the same algorithm will work replacing \( p \) with \( q \) and omitting Step 2.
- For \( f \in \mathbb{Q} \), choose the smallest \( b \in \mathbb{N} \) such that \( b f \in \mathbb{Z} \).
- Then \( b \) is a perfect power if \( f \), so we run the algorithm on input \( b f \).
- For \( f \in \mathbb{R}[x_1, \ldots, x_n] \), choose random values \( b_1, \ldots, b_n \in \mathbb{R} \). Then test whether \( \forall b_1, \ldots, b_n \in \mathbb{R} \), \( f \) is a perfect power.

5 Why it works

Reduction: Since \( p \mid \text{disc}(f) \), \( f \) is an \( r \)-th power over \( \mathbb{Z}[x] \).

Detection: \( f(a_1, \ldots, a_n)^{r-1} = 1 \) iff \( f(a_n) \) is an \( r \)-th power in \( \mathbb{F}_q \).

Implication: Clearly, if \( f = h^r \) for some \( h \), \( r \geq 2 \), then each \( f(a) \) is a perfect \( r \)-th power in \( \mathbb{F}_q \).

The other direction is more interesting:

Theorem. Suppose \( f \in \mathbb{F}_q[x] \), \( f \) is not a perfect \( r \)-th power, and the degree of \( f \) is not more than \( 1 + \sqrt{2} \).

Then, for a random \( c \in \mathbb{F}_q \), the probability that \( f(a) \) is a perfect \( r \)-th power in \( \mathbb{F}_q \) is less than \( 3/4 \).

The proof uses an exponential character sum argument and the powerful Weil’s Theorem for character sums with polynomial arguments. Since \( (1/4)^{\sqrt{3/4}} \leq 1/4 \), choosing 5 random evaluations guarantees success with at least 3/4 probability.

6 How high can the power be?

- Speed of algorithm depends on how many \( r \)'s and how big.
- Number of \( r \)’s is \( O(\log \deg f) \) since all are distinct prime divisors of \( \deg f \).
- But can \( r \) be large?

7 Complexity

How big is \( p \)?

\[
disc(f) = \text{res}(f, f') \in \mathbb{O}(\log(n) + \log \| f \|_1).
\]

A prime with \( O(\log(n) + \log \| f \|_1) \) bits does not divide the discriminant with high probability.

How many operations in \( \mathbb{F}_q \)?

The most costly step is computing \( f(a_1, \ldots, a_n) \) at Step 3. Computing each \( f(a_i) \) can be accomplished using \( O(\log n) \) operations in \( \mathbb{F}_q \), and these can be raised to the power \( (q - 1)/r \) with an additional \( O(\log q - \log r) \) operations in \( \mathbb{F}_q \).

How costly are operations in \( \mathbb{F}_q \)?

If we choose \( q = p^r \) and work in a field extension modulo an irreducible polynomial in \( \mathbb{F}_q \) of degree \( r - 1 \), then each operation in \( \mathbb{F}_q \) will cost \( \mathcal{O}(r^2 \log^2 q) \) bit operations, which is \( \mathcal{O}(r^2 \log(q + \log \| f \|_1)) \).

Total Bit Complexity: Finally, because \( r \in O(\log(\| f \|_1)) \), we have a total bit complexity of \( O((\log q) \| f \|_1 \log n) \), which is polynomial in the lacunary size of \( f \), as desired.

8 Implementation

We used Victor Shoup’s NTL to implement and compare the performance of the following three algorithms. This is a C++ library which, when compiled with GMP, provides (probably) the fastest implementations of arithmetic for dense univariate polynomials, multivariate polynomials, and integers, and finite fields.

Reference


