AN IMPROVED ALGORITHM FOR DIOPHANTINE EQUATIONS IN ONE VARIABLE

ADAM STRZEBOŃSKI

ABSTRACT. We present a new algorithm for computing integer roots of univariate polynomials. For a polynomial f in $Z[t]$ we can find the set of its integer roots in a time polynomial in the size of the sparse encoding of f. An algorithm to do this was given in [1]. The paper introduces a polynomial time method for computing sign of f at integral points, and then finds integer roots of f by isolating the real roots of all polynomials in the sparse derivative sequence of f, up to unit-length intervals. We propose to isolate the roots of f using a "sparse variant" of Fourier's theorem, and show that our algorithm requires a smaller number of polynomial sign evaluations. We present an empirical comparison of our implementations of the original and of the improved algorithm, and of an algorithm using modular root finding and Hensel lifting, suitable for dense polynomials.

1. INTRODUCTION

Let

$$f = a_1 t^{e_1} + \ldots + a_k t^{e_k}$$

where $e_1 > \ldots > e_k$, and $a_i \in Z \setminus \{0\}$, for $1 \leq i \leq k$, and let us denote

$$minexp(f) := e_k$$

Definition 1.1. The sparse derivative sequence of f, is the sequence $f_1, \ldots, f_k$ defined by

$$f_1 := f/t^{minexp(f)}$$

and

$$f_{i+1} := f_i'/t^{minexp(f_i')}$$

for $1 \leq i \leq k-1$.

The algorithm described in [1] finds integer roots of f by isolating the real roots of polynomials of the sparse derivative sequence of f, taken in the reverse order, up to intervals $(u, u+1)$ or $[u,u]$, with $u \in Z$. The method is based on the fact that, by Rolle's theorem, a polynomial can have at most one root between two subsequent roots of its derivative. In the worst case the algorithm may require isolating k^2 roots.

Our algorithm isolates the roots of f directly, using a "sparse variant" of Fourier's theorem proven in the next section. It requires isolating at most $2k-2$ roots of f or polynomials in its sparse derivative sequence. In Section 3 we present the algorithm. Next, we give a more detailed comparison of its complexity with the complexity of the algorithm given in [1].

In Section 5 we present an algorithm finding roots of f modulo a prime and using Hensel lifting to obtain the integer roots. Since f may have e_1 roots modulo a prime p, even if $k = 2$, (for instance $f = t^p - t$), this algorithm is exponential in the size of f. However, our experiments suggest that for dense polynomials this method is the fastest.

Date: December 21, 2001.

The last section contains empirical comparison of our implementations of the three algorithms.

2. Root isolation theorem

Let $f_1, \ldots, f_k : I \to R$ be a sequence of differentiable functions defined in an open interval $I \subseteq R$, and such that for all $1 \leq i \leq k-1$, and for all $x \in I$

$$sign(f_{i+1}(x)) = sign(f_i'(x)) \tag{2.1}$$

and

$$f_k(x) \neq 0$$

For $x \in I$, and $1 \leq i \leq j \leq k$, let $sgc_{i,j}(x)$ denote the number of sign changes in the sequence $f_i(x), \ldots, f_j(x)$ with terms equal to zero removed, and let $sgc(x) := sgc_{1,k}(x)$.

Theorem 2.1. *For any $a \in I$*

$$\lim_{x \to a^+} sgc(x) = sgc(a)$$

and

$$\lim_{x \to a^-} sgc(x) = sgc(a) + r + 2s$$

where $r \in Z_+$, $s \in Z_+ \cup \{0\}$, and

$$f_1(a) = \ldots = f_r(a) = 0,\ f_{r+1}(a) \neq 0$$

Moreover, $s = 0$ unless there is $t > r+1$ such that $f_t(a) = 0$.

Proof. Note that if

$$i_0 = 1 < i_1 < \ldots < i_m < i_{m+1} = k$$

then

$$sgc(x) = \sum_{l=0}^{m} sgc_{i_l, i_{l+1}}(x) \tag{2.2}$$

If $f_l(a) \neq 0$ for all $i \leq l \leq j$, then clearly $sgc_{i,j}$ is constant in a neigbourhood of a.

Now suppose that

$$f_p(a) = \ldots = f_q(a) = 0,\ f_{q+1}(a) \neq 0$$

Conditions 2.1 imply that there is a neighbourhood U of a in which none of f_i , $p \leq i \leq q+1$, has zeroes other than a, and we have one of the following four sign combinations uniquely determined by the signs of f_p in U.

	$x<a$	a	$x>a$	$x<a$	a	$x>a$
f_p	$-$	0	$+$	$+$	0	$-$
f_{p+1}	$+$	0	$+$	$-$	0	$-$
f_{p+2}	$-$	0	$+$	$+$	0	$-$
$\ldots$						
f_q	$-$	0	$+$	$+$	0	$-$
f_{q+1}	$+$	$+$	$+$	$-$	$-$	$-$

	$x<a$	a	$x>a$	$x<a$	a	$x>a$
f_p	$+$	0	$+$	$-$	0	$-$
f_{p+1}	$-$	0	$+$	$+$	0	$-$
f_{p+2}	$+$	0	$+$	$-$	0	$-$
$\ldots$						
f_q	$-$	0	$+$	$+$	0	$-$
f_{q+1}	$+$	$+$	$+$	$-$	$-$	$-$

If $p = 1$, then, for $x \in U$, $sgc_{1,q+1}(x) = q$ if $x < a$ and $sgc_{1,q+1}(x) = 0$ if $x \geq a$.

If $p > 1$, and $f_{p-1}(a) \neq 0$, then there is a neigbourhood $V \subseteq U$ of a, such that $f_{p-1}(x) \neq 0$ for $x \in V$. If $x \in V$, and we have one of the first two sign combinations, then $q-p$ is even and, depending on the sign of f_{p-1} in V, either

$$sgc_{p-1,q+1}(x) = \begin{cases} q-p+2, & for x < a \\ 0, & for x \geq a \end{cases}$$

or

$$sgc_{p-1,q+1}(x) = \begin{cases} q-p+1, & for x < a \\ 1, & for x \geq a \end{cases}$$

If $x \in V$, and we have one of the last two sign combinations, then $q-p$ is odd and, depending on the sign of f_{p-1} in V, either

$$sgc_{p-1,q+1}(x) = \begin{cases} q-p+2, & for x < a \\ 1, & for x \geq a \end{cases}$$

or

$$sgc_{p-1,q+1}(x) = \begin{cases} q-p+1, & for x < a \\ 0, & for x \geq a \end{cases}$$

The discussion above combined with equation 2.2 proves the theorem. □

Corollary 2.2. *Let $f_1, \ldots, f_k$ be the sparse derivative sequence of a polynomial f, let $0 \leq a < b$, and let $1 \leq p \leq k$, be such that $sgc_{p,k}(a) = sgc_{p,k}(b)$. Then*

(1) *f_p has a constant nonzero sign on $(a,b]$.*
(2) *$sgc_{p,k}(c) = sgc_{p,k}(a)$ for any $a \leq c \leq b$.*
(3) *$sgc_{1,p}(a) - sgc_{1,p}(b) = r + 2s$, where r is the number of roots of f in $(a,b]$, counted with multiplicities, and s is a nonnegative integer. In particular, if $sgc_{1,p}(a) = sgc_{1,p}(b)$ then f has no roots in $(a,b]$, and if $sgc_{1,p}(a) - sgc_{1,p}(b) = 1$ then f has exactly one root in $(a,b]$ and $sign(f(a)) \neq sign(f(b))$.*

Corollary 2.3. *A polynomial with $k > 0$ nonzero terms has at most $k-1$ positive roots and at most $k-1$ negative roots, and hence at most $2k-1$ real roots. For completeness, let us note that the polynomial*

$$f = \prod_{i=0}^{k-1} (t^2 - i^2)$$

has exactly k nonzero terms and $2k-1$ real roots.

3. The algorithm

We present an algorithm computing the positive integer roots of a polynomial $f \in Z[t] \setminus \{0\}$. To find the negative integer roots of f we compute the positive integer roots of $f(-t)$. To compute the sign of a polynomial we use the algorithm described in [1]. As a positive integer root bound for f, we take the smaller of the bound used in [1] (absolute value of the trailing coefficient of f), and the positive root bound described in [2] (Theorem 9). We also use the following two algorithms. *BinarySearch*(f,a,b), for $f \in Z[t] \setminus \{0\}$ and $a,b \in Z$, assuming that f has exactly one root in (a,b), and $f(a)f(b) < 0$, finds out whether the root is an integer, and if yes returns it. The algorithm uses binary search with integer subdivision points, and stops when it finds an integer root, or isolates the root to an interval of length 1. *ExhaustiveSearch*(f,a,b,r), for $f \in Z[t] \setminus \{0\}$ and $a,b \in Z$, finds the integer roots of f in $[a,b]$, assuming that there is at most r of them. The algorithm computes the sign of f at subsequent integers in $[a,b]$, and stops when it finds r roots, or when it has checked all integers in $[a,b]$.

Algorithm 3.1. *Given $f \in Z[t] \setminus \{0\}$, returns the list of positive integer roots of f.*

(1) *Compute the sparse derivative sequence $f_1, \ldots, f_k$ of f, a positive integer root bound rb for f_1, and set $rootlist$ to an empty list. While $f_1(rb) = 0$ and $rb > 0$ add rb to $rootlist$ and decrement rb. Put interval $(0, rb)$ on $intervalstack$.*
(2) *If $intervalstack$ is empty, return $rootlist$, else take an interval (a,b) off $intervalstack$.*
(3) *If $sgc(a) = sgc(b)$ there are no roots of f in (a,b). Go to step 2.*
(4) *If $sgc(a) - sgc(b) = 1$, compute $BinarySearch(f_1, a, b)$. If it finds an integer root add it to $rootlist$. Go to step 2.*
(5) *If $b - a \leq k$, compute $ExhaustiveSearch(f_1, a, b, sgc(a) - sgc(b))$. Add the roots found to $rootlist$. Go to step 2.*
(6) *Choose integers $a < c = d < b$. While $f_1(c) = 0$ and $c > a$ add c to $rootlist$ and decrement c. While $f_1(d) = 0$ and $d < b$ add d to $rootlist$ and increment d. Put intervals (a,c) and (d,b) on $intervalstack$, and go to step 2.*

Remark 3.2. There are several efficiency improvents that can be made to the algorithm described above.

(1) *$sgc(0)$ is equal to the number of sign changes in the coefficient list of f. We can compute it before computing the sparse derivative sequence, and if it is 0 or 1, we know that there are no roots, or we can find the only root using BinarySearch. Similarly, if rb is small ($rb \leq k$ in our implementation), we use $ExhaustiveSearch$ on $[1, rb]$, without computing the sparse derivative sequence.*
(2) *We can construct the loop somewhat differently, so that we check whether $sgc(a) - sgc(b)$ is 0 or 1, or whether $b - a$ is small before adding (a,b) to $intervalstack$. This way $intervalstack$ contains only intervals for which $sgc(a) - sgc(b) > 1$, and hence its height never exceeds $(k-1)/2$.*
(3) *Keep the computed signs of $f_1, \ldots, f_k$ at interval's endpoints together with the interval, so that we don't need to recompute them.*
(4) *For every interval (a,b) which we need to subdivide, compute the smallest p satisfying the conditions of 2.2. Then for any point c subdividing (a,b) we only need to compute the signs of $f_1(c), \ldots, f_{p-1}(c)$.*
(5) *In point 6 of the algorithm we choose $c = floor((a+b)/2)$, except that since the changes in value of sgc tend to happen close to zero, for intervals with $a = 0$ we choose c to be closer to zero than $b/2$.*

4. Complexity analysis

Let us compute the number of polynomial sign evaluations required by our algorithm (let us call it FIR, for Fourier Integer Roots), and by the algorithm described in [1] (let us call it RIR for Rolle Integer Roots). Let $f \in Z[t] \setminus \{0\}$, let k be the number of nonzero terms in f, and let M be a bound for integer roots of polynomials in the sparse derivative sequence $f_1, \ldots, f_k$ of f. For simplicity we take a common root bound for all $f_1, \ldots, f_k$. Since the worst case estimate for size of coefficients of polynomials f_i grows with i, taking a common estimate is "to the advantage" of RIR.

FIR isolates at most $2k-2$ changes in value of sgc, from an interval of length of at most M to an interval of length of at least 1, evaluating signs of at most k polynomials for each interval subdivision. Hence, the total number of polynomial sign evaluations is bounded by $O(k^2 logM)$.

To isolate roots of f_{k-m+1} RIR needs to evaluate f_{k-m+1} at at most $(m-1)^2$ endpoints of intervals isolating roots of $f_{k-m+2}, \ldots, f_k$, and then it needs to isolate at most $2m-1$ roots

of f_{k-m+1} from an interval of length of at most M to an interval of length 1, evaluating 1 polynomial sign for each subdivision. Hence, the number of polynomial sign evaluations necessary for isolating roots of f_{k-m+1}, assuming the roots of $f_{k-m+2},\ldots,f_k$ have already been isolated, is bounded by $O(m^2+mlogM)$. Therefore the total number of polynomial sign evaluations necessary for finding the integer roots of f using RIR is bounded by $O(k^3+k^2logM)$.

Clearly when $k \gg logM$ FIR has a lower complexity than RIR. However, the complexity computed above is a worst case estimate. In practice we quite often get large intervals with $sgc(a)-sgc(b)=1$, in which case in the further subdivision of the interval we need only to compute signs of f_1. Also, the use of Remark 3.2 helps to lower the number of polynomial signs that need to be computed with each interval subdivision.

5. THE DENSE POLYNOMIAL CASE

Another possible approach to finding integer roots of a polynomial f is to find the roots of f modulo a prime p, and then use Hensel lifting to find the integer roots. Such an algorithm is exponential in the size of f because the number of roots modulo p may be as large as the degree of f. Therefore the algorithm is not suitable for sparse polynomials. However, it works quite well when f is dense.

Algorithm 5.1. *Given $f \in Z[t]\setminus\{0\}$, returns the list of integer roots of f.*

(1) *Compute the square free part g of f, and an integer root bound rb for g.*
(2) *Select a prime p such that g is square free modulo p.*
(3) *Compute roots $r_1,\ldots,r_s$ of g modulo p, by computing $h=gcd(g,t^p-t)$ and factoring it using the algorithm described in* [3]*, section 18.3.*
(4) *Lift $r_1,\ldots,r_s$ to roots $r'_1,\ldots,r'_s$ of g mod p^{2^m}, with $p^{2^m}\geq 2rb+1$, using the quadratic iteration Hensel lifting described in* [3]*, section 16.2.*
(5) *Return those of $r'_1-p^{2^m},\ldots,r'_s-p^{2^m},r'_1,\ldots,r'_s$ which are in the interval $[-rb,rb]$ and are roots of g.*

The integer root bound is the greater of the positive and negative integer root bounds computed as in Section 3. In our implementation whenever the root bound is less than 10 we find the integer roots using a direct search. In step 2 we try an increasing sequence of primes, until we find one for which g is square free modulo p. Primes for which g is not square free modulo p divide the discriminant of g, therefore there is only a finite number of them.

6. EXPERIMENTAL RESULTS

In this section we present an empirical comparison of our implementations of Algorithm 3.1 (FIR, for Fourier Integer Roots), Algorithm 5.1 (HIR, for Hensel Integer Roots), and of the algorithm described in [1] (RIR, for Rolle Integer Roots). Both FIR and RIR use the refinement proposed in section 4 of [1]. All three algorithms have been implemented as a part of *Mathematica* kernel. The computations have been done on a 700 MHz Pentium III laptop computer with 256 MB of RAM. Time is given in seconds, columns marked st# show the number of polynomial sign tests used in FIR and RIR. The results given are averages for sets of 10 randomly generated polynomials.

We have used the following three types of randomly generated polynomials.

TABLE 1. Random sparse polynomials

Polynomial	Deg	Terms	Roots	Time HIR	Time FIR	Time RIR	st# FIR	st# RIR
$SP(10^3,1,10)$	10^3	19.9	1	1.1	0.005	0.005	110	317
$SP(10^3,2,10)$	10^3	29.2	2	1.11	0.011	0.085	274	3616
$SP(10^3,4,10)$	10^3	49.2	4	1.14	0.055	0.96	799	27299
$SP(10^3,8,10)$	10^3	89.3	8	1.19	0.22	7.21	2198	122415
$SP(10^3,16,10)$	10^3	157.1	16	1.31	0.88	65.6	5196	575958
$SP(10^3,32,10)$	10^3	294.9	32	1.64	4.72	>300	12825	?
$SP(10^2,4,10)$	10^2	43.1	4	0.028	0.034	0.68	722	22672
$SP(10^4,4,10)$	10^4	50	4	40.4	0.33	0.38	456	2874
$SP(10^5,4,10)$	10^5	50	4	>300	5.61	5.65	418	1031
$SP(10^6,4,10)$	10^6	50	4	>300	76.5	76.5	420	1031

TABLE 2. Random dense polynomials

Polynomial	Deg	Roots	Time HIR	Time FIR	Time RIR	st# FIR	st# RIR
$DP(128,128)$	128	128	1.09	28.3	>300	22372	?
$DP(128,32)$	128	32	0.23	1.33	130	6548	734241
$DP(128,8)$	128	8	0.086	0.26	12.2	2337	218165
$DP(128,2)$	128	2	0.052	0.13	2.1	879	58697
$DP(128,0)$	128	0	0.008	0.015	0.11	82	545

(1) Random sparse polynomials of degree n with k integer roots.

$$SP(n,k,p) = (a_1 t^{n-k} + \sum_{i=2}^{p-1} a_i t^{e_i} + a_p) \prod_{j=1}^{k} (t - r_j)$$

where a_i and r_j are randomly generated 100-bit integers, r_j are all distinct, and $0 < e_i < n-k$ are randomly generated distinct exponents.

(2) Random dense polynomials of degree n with k integer roots.

$$DP(n,k) = (\sum_{i=0}^{n-k} a_i t^i) \prod_{j=1}^{k} (t - r_j)$$

where a_i and r_j are randomly generated 100-bit integers, r_j are all distinct.

(3) Sparse polynomials with $k+1$ terms and the maximal number $2k+1$ of integer roots.

$$PP(k,p) = t \prod_{j=1}^{k} (t^p - r_j^p)$$

where p is an even integer, and r_j are randomly generated distinct 10-bit integers.

For all random sparse polynomials we tried FIR was the fastest algorithm, except for polynomials with the highest density (29 % and 43 %), where HIR was faster. The speed advantage of FIR over RIR grows with the number of terms. For very high degree polynomials we see that the number of polynomial sign tests performed both by FIR and RIR decreases. This is a result of using the refinement proposed in section 4 of [1].

TABLE 3. Sparse polynomials with the maximal number of integer roots

Polynomial	Deg	Terms	Roots	Time HIR	Time FIR	Time RIR	st# FIR	st# RIR
$PP(5,2)$	10^1	6	10	0.003	0.003	0.007	139	337
$PP(5,20)$	10^2	6	10	0.006	0.007	0.012	128	314
$PP(5,200)$	10^3	6	10	0.018	0.018	0.034	126	268
$PP(5,2000)$	10^4	6	10	16.3	0.46	0.51	129	142
$PP(5,20000)$	10^5	6	10	>300	14.5	16.3	126	142

When the polynomials are dense, HIR is clearly the fastest method. FIR is slower, but still much faster than RIR.

For random sparse polynomials with the maximal number of roots the fastest algorithm is FIR. HIR gives the same timings as FIR for smaller degree polynomials, but for higher degrees it becomes much slower. RIR is slower than FIR by a factor which decreases with the degree.

7. CONCLUSIONS

We have presented two algorithms for solving Diophantine equations in one variable. Our algorithm based on a "sparse" version of Fourier's theorem is suitable for sparse polynomials, and is consistently faster than the algorithm described in [1]. The second algorithm, based on modular root finding and Hensel lifting is the best choice for dense polynomials.

REFERENCES

[1] F. Cucker, P. Koiran, S. Smale, "A Polynomial Time Algorithm for Diophantine Equations in One Variable", J. Symbolic Comp., 27 (1999), 21–29.

[2] J. R. Johnson, "Algorithms for Polynomial Real Root Isolation", in B. Caviness, J. Johnson (eds.), Quantifier Elimination and Cylindrical Algebraic Decomposition, Springer-Verlag 1998, 269–299.

[3] R. Zippel, "Effective Polynomial Computation", Kluwer Academic Publishers, Boston/Dodrecht/London 1993.

WOLFRAM RESEARCH INC., 100 TRADE CENTRE DRIVE, CHAMPAIGN, IL 61820, U.S.A.
E-mail address: adams@wolfram.com