Real-Root Isolation of Polynomials

XINLONG YI,
Computer Science and Engineering Department, University of California Riverside, USA

1 ABSTRACT

Computing the real roots of univariate polynomial is one of fundamental tasks in numeric and computer algebra. Usual root-finding algorithms for computing the real roots of a polynomial may produce some real roots, but cannot generally certify having found all real roots.

In this project, we implemented a basic real roots isolation program based on Budan's Theorem and Continued Fraction Method. Both methods are developed from Descartes' rule of signs. Then we compare the running time of these two methods and get the conclusion that TODO.

2 INTRODUCTION

One of the most fundamental scientific computation is to computing the real roots of polynomials. For the polynomials with low order, like quadratic or cubic, we can using formula to get roots directly. However, according to the Abel–Ruffini theorem[1], there is no solution in radicals to general polynomial equations of degree of five or higher with arbitrary coefficients. Therefore, general root-finding algorithms are needed for general polynomials.

However, the usual root-finding algorithms, like Newton Method, cannot generally certify having found all real roots. Especially, if such algorithms does not find any root, one cannot know if there is real roots or not. In order to get all real roots, real-root isolation is useful. Real-root isolation can generate intervals, which contain only one real root of the polynomial, so that no real root will be missed.

3 LITERATURE REVIEW

4 METHODOLOGY

In this section, I will introduce the methods that applied to this project. Since both two methods are only work on square-free polynomials, the first step of this project is appling square free decomposition to original polynomials to avoid repeat roots. After square free decomposition, methods based on Budan's Theorem and Continued Fraction will be applied to square-free polynomials. They both based on Descartes' rule of sign to check how many real roots in a interval. The continued fraction method also uses Mobius transformation, which will be introduce in Continued Fraction subsection.

4.1 Square Free Decomposition

In mathematics, a square-free polynomial is a polynomial defined over a field that does not have a divisor any square of a non-constant polynomial[2]. Usually, a square-free polynomial refers to the polynomials with no repeated roots. This project applied Yun's algorithm[2] to perform square-free decomposition. It's based one the succession of Greatest Common Divisor(GCD).

4.1.1 Greatest Common Divisor. In algebra, the greatest common divisor of two polynomials is a polynomial, of the highest possible degree, that is a factor of both the two original polynomials. This concept is simillar to the GCD of two integers.

Author's address: Xinlong Yi, xyi007@ucr.edu,
Computer Science and Engineering Department, University of California Riverside, 900 University Ave, Riverside, California, USA, 92507.

This project applied Euclid's algorithm to compute the GCD of two polynomials. In the Algorithm 1, $rem(a, b)$ refers to the remainder of Euclidean division of polynomial a and polynomial b.

Algorithm 1: GCD of two polynomials

input : $P1$: a univariate polynomial
$P2$: a univariate polynomial
output : Greatest common divisor of $P1$ and $P2$
$r_0 = P1$;
$r_1 = P2$;
for $i = 1; r_i \neq 0; i = i + 1$ **do**
  $r_{i+1} = rem(r_{i-1}, r_i)$
end
return r_{i-1}

4.1.2 Yun's Algorithm. Based on the succession of GCD, Yun developed a square-free decomposition algorithm for univariate polynomials. Given a primitive polynomial P, the algorithm will compute square-free polynomials P_i and the subscript refers to the times this square-free polynomial appears. Which means $P = \prod_{i=1}^{k} P_i^i$.

Algorithm 2: Yun's Square-free Decomposition Algorithm

input : P: primitive polynomial
output : List of square-free polynomials
$G = GCD(P, dP/dx)$;
$C_1 = P/G$;
$D_1 = (dP/dx)/G - dC_1/dx$;
for $i = 1, C_i \neq 0; i = i + 1$ **do**
  $P_i = GCD(C_i, D_i)$;
  $C_{i+1} = C_i/P_i$;
  $D_{i+1} = D_i/P_i - dC_{i+1}/dx$;
end
return $P_1, P_2 ... P_k$

4.2 Budan's Theorem

Budan's Theorem is a theorem used for bounding the numnber of real roots in a given interval. Given a univariate polynomial P, we denote $\#_{l,r}(P)$ as the number of real roots of P in half-open interval $(l, r]$. Then we denote $v_h(P)$ as the number of sign changes in coefficients of polynomial P_h, where $P_h(x) = P(x + h)$.

Budan's Theorem states that $v_l(h) - v_r(h) - \#_{l,r}(P)$ is a nonnegative even integer.

From above statement, we can know that if $v_l(h) - v_r(h) = 1$ or 0, there is only one real root or no root in interval $(l, r]$.

Based on this theorem, this project applied bisection to isolate the real roots. This process described in Algorithm 3

Algorithm 3: Real-root isolation based on Budan's Theorem

input : P square-free polynomial
output: List of intervals contains only one real root
$ret = []$;
$up\_bound = Upper(P)$;
$low\_bound = -up\_bound$;
$search = [(low\_bound, up\_bound)]$;
while *search not empty* **do**
  $l, r = pop(search)$;
  $vl = sign\_change(P(x + l))$;
  $vr = sign\_change(P(x + r))$;
  if $vl - vr = 1$ **then**
    $ret.append([l, r])$;
  end
  else if $vl - vr > 1$ **then**
    $mid = l + (r - l)/2$;
    $search.append(mid, r)$;
    $search.append(l, mid)$;
  end
end
return *ret*;

In Algorithm 3, $Upper(P)$ returns the upper bound of real roots of P. We are using Lagrange's bound in this project. Assuming $P = a_0 + a_1x + ... + a_nx^n$, Lagrange's bound is $max\{1, \sum_{i=0}^{n-1} |\frac{a_i}{a_n}|\}$.

4.3 Continued Fraction

Let's first introduce some notation used in this algorithm. Let $M(x)$ represent a Mobius Transformation, which map x to $\frac{ax+b}{cx+d}$. So that the number of positive roots of $P(M(x))$ equals to the number of roots in interval $(\frac{b}{d}, \frac{a}{c}]$. And $s = sign\_change(P)$ represents the sign changes along coefficients of P.

We using $\{a, b, c, d, p, s\}$ to represent a interval. Where, $ad - bc \neq 0$ and the roots of original polynomial P in $(\frac{b}{d}, \frac{a}{c}]$ are images of positive roots of p. $s = sign\_change(p)$.

The algorithm can be described as Algorithm 4.

Algorithm 4: Real-root isolation based on Continued Fraction

input : P square-free polynomial with no zero root
output: List of intervals contains only one positive real root
$s = sign\_change(P)$;
if $s = 0$ **then**
  return *[]*;
end
else if $s = 1$ **then**
  return $[(0, \infty)]$;
end
$ret = []$;
$intervals = [\{1, 0, 0, 1, P, s\}]$;
while *intervals not empty* **do**
  $\{a, b, c, d, p, s\} = pop(intervals)$;
  $p' = p(x + 1)$;
  if $p'(0) = 0$ **then**
    $ret.append([\frac{b}{d}, \frac{b}{d}])$;
    $p' = p'/x$;
  end
  $s' = sign\_change(p')$;
  $intervals.append(\{a, a + b, c, c + d, p', s'\})$;
  if $s - s' = 1$ **then**
    $ret.append([\frac{a}{c}, \frac{b}{d}])$;
  end
  else if $s - s' > 1$ **then**
    $p'' = p(b/(1 + x))$;
    $intervals.append(\{b, a + b, d, c + d, p'', sign\_change(p'')\})$;
  end
end
return *ret*;

Above algorithm only accept the polynomials with non-zero roots, therefore before using the Algorithm 4, we need to remove the zero roots of original P. Then we used Algorithm 4 with $P(x)$ and $P(-x)$ to get the positive roots and negative roots.

5 IMPLEMENTATION

This section will

6 CONCLUSION

REFERENCES

[1] Raymond G. Ayoub. 1980. Paolo Ruffini's Contributions to the Quintic. *Archive for History of Exact Sciences* 23, 3 (1980), 253–277. http://www.jstor.org/stable/41133596

[2] David Y.Y. Yun. 1976. On Square-Free Decomposition Algorithms. In *Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation* (Yorktown Heights, New York, USA) *(SYMSAC '76)*. Association for Computing Machinery, New York, NY, USA, 26–35. https://doi.org/10.1145/800205.806320