A Fast & Robust Solution for Cubic & Higher-Order Polynomials

Cem Yuksel
University of Utah
Salt Lake City, Utah, USA
cem@cemyuksel.com

ABSTRACT

We present a computationally-efficient and numerically-robust method for finding real roots of cubic and higher-order polynomials. It begins with determining the intervals where a given polynomial is monotonic. Then, the existence of a real root within each interval can be quickly identified. If one exists, we find the root using a stable variant of Newton iterations, providing fast and guaranteed convergence and satisfying the given error bound.

ACM Reference Format:
Cem Yuksel. 2022. A Fast & Robust Solution for Cubic & Higher-Order Polynomials. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH '22 Talks), August 07-11, 2022.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3532836.3536266

1 INTRODUCTION

Polynomials can be found everywhere in computer graphics and often times we are interested in finding their real roots. However, an efficient analytical solution exists only for quadratic (second order) polynomials. Even for cubic (third order) polynomials, we must resort to numerical methods, because the analytical solution is not efficient or robust enough. Higher-order polynomials are often avoided, mainly because of the common misconception that they would be too expensive to solve. Below, we correct this by presenting an efficient solution, starting with quadratics, then discussing cubics, and finally extending it to higher-order polynomials.

2 QUADRATIC POLYNOMIALS

To find the roots of a quadratic polynomial $ax^2 + bx + c = 0$, we recommend using a variant of the commonly-known formula

$$x_1 = -\frac{2c}{b + (\operatorname{sgn} b)\sqrt{\Delta}} \qquad x_2 = -\frac{b + (\operatorname{sgn} b)\sqrt{\Delta}}{2a}\,, \tag{1}$$

where $\Delta = b^2 - 4ac$. This version is more stable than the common form, because it avoids evaluating the difference in $b \pm \sqrt{\Delta}$. Also, it works even when $a = 0$, producing $x_1 = -c/b$ and $x_2 = \pm\infty$.

3 CUBIC POLYNOMIALS

While there exists a formula for computing the real roots of cubic polynomials, it involves computationally-expensive cubic root operations and a direct implementation may lead to excessive precision loss. The numerical solution we present below is 2.5× to 6× faster in our tests, provides better accuracy, and can be further accelerated when an approximate solution is sufficient.

3.1 Splitting Cubic Polynomials

We begin with splitting the given cubic polynomial $f(x)$ into three monotonic intervals using its 2 critical points x_1 and x_2, where its derivative $f'(x)$ is zero. These critical points can be easily evaluated, as they are the roots of the quadratic polynomial $f'(x)$. In-between and beyond these critical points, the cubic polynomial $f(x)$ either monotonically increases or decreases.

For any monotonic interval $x \in [x_1, x_2]$, there exists a single real root if and only if exactly one of $f(x_1)$ and $f(x_2)$ is negative. Otherwise, $f(x)$ cannot go through zero within $x \in [x_1, x_2]$. Thus, we can quickly check the existence of a root by evaluating $f(x_1)$ and $f(x_2)$, and comparing their signs. For infinite intervals $(-\infty, x_1]$ and $[x_2, \infty)$, the cubic term's sign determines the sign of $f(\pm\infty)$.

Most applications, however, would need roots within a given interval $[x_{\text{start}}, x_{\text{end}}]$. Depending on whether the critical points x_1 and x_2 are within $[x_{\text{start}}, x_{\text{end}}]$, we split the given interval up to 3 pieces. For each piece, if a root exists within its interval, we perform numerical root finding to evaluate it, as explained below.

3.2 Numerical Root Finding for Cubics

Newton iterations provide the fastest numerical method for root finding. Starting with a guess $x_r^{(0)}$, at each iteration j we compute the next guess using

$$x_n^{(j+1)} = x_r^{(j)} - \frac{f(x_r^{(j)})}{f'(x_r^{(j)})}\,. \tag{2}$$

Yet, Newton iterations are not always stable, particularly when $|f'(x_r^{(j)})|$ is small. For cubics, this happens near the critical points. Thus, the resulting guess $x_n^{(j+1)}$ might be arbitrarily far from the root, possibly even more so than the previous guess $x_r^{(j)}$.

One simple way to ensure stability would be forcing the next guess to remain within the given interval $[x_{\min}, x_{\max}]$, using

$$x_r^{(j+1)} = \max\left(x_{\min}, \min\left(x_{\max}, x_n^{(j+1)}\right)\right)\,. \tag{3}$$

Indeed, this simple step solves the convergence problems for cubics, except for one special case when

$$\frac{f(x_r^{(j+1)})}{f'(x_r^{(j+1)})} = -\frac{f(x_r^{(j)})}{f'(x_r^{(j)})}\,, \tag{4}$$

which produces a next guess $x_r^{(j+2)} = x_r^{(j)}$, resulting in an infinite loop. Fortunately, this can only happen for the interval within the two critical points $[x_1, x_2]$ and only when two consecutive guesses are on either side of the inflection point x_c, which is exactly at the center of the two critical points $x_c = (x_1 + x_2)/2$, where the second derivative changes sign. Therefore, we can easily avoid this

SIGGRAPH '22 Talks, August 07-11, 2022, Vancouver, BC, Canada

case by splitting the interval between the two critical points into two intervals $[x_1, x_c]$ and $[x_c, x_2]$. Note that only one of them can contain a root and the other can be safely discarded.

When the step size $|x_r^{(j+1)} - x_r^{(j)}|$ is below a desired error bound ϵ, the iterations can be terminated by returning the latest guess. Because these iterations are guaranteed to converge, there is no need to limit the number of iterations, though a fixed number can be used when performance is more important than accuracy.

4 HIGHER-ORDER POLYNOMIALS

This approach can be easily extended to higher-order polynomials.

4.1 Splitting Higher-Order Polynomials

Similar to the cubic case, for a polynomial of degree d, we first find the critical points by solving for the roots of its derivative, which is a polynomial of degree $d-1$. This results in a recursive algorithm.

Then, we split the given interval $[x_{\text{start}}, x_{\text{end}}]$ into smaller pieces, using those critical points that are within this interval, if any. For each piece, we check the value of the polynomial $f(x)$ at its end points. If these values have opposing signs (i.e one positive and the other negative), we proceed to numerical root finding to evaluate the real root within this piece. Otherwise, we discard the piece.

4.2 Generalized Numerical Root Finding

Again, Newton iterations can be used for quickly finding the real root within each interval. However, ensuring stability of Newton iterations with higher-order polynomials is not as simple. We employ two strategies for guaranteeing convergence.

First, we iteratively shorten the interval. Starting with the initial interval $[x_{\text{min}}^{(0)}, x_{\text{max}}^{(0)}]$, at each iteration we update it by splitting the interval at the previous guess. Since we compute f at each iteration, we can shorten the interval with minimal additional work, using

$$\begin{aligned} x_{\text{min}}^{(j+1)} &= x_r^{(j)}, \quad \text{if } \operatorname{sgn} f(x_{\text{min}}^{(j)}) = \operatorname{sgn} f(x_r^{(j)}) \\ x_{\text{max}}^{(j+1)} &= x_r^{(j)}, \quad \text{if } \operatorname{sgn} f(x_{\text{max}}^{(j)}) = \operatorname{sgn} f(x_r^{(j)}) \ . \end{aligned} \tag{5}$$

While shortening the interval helps, it does not guarantee convergence. This is because Newton iterations can still bounce between the two ends of the interval without ever shortening it. To avoid this and ensure convergence, we fall back on bisection whenever Newton iteration fails to produce a next guess inside the interval:

$$x_r^{(j+1)} = \begin{cases} x_n^{(j+1)}, & \text{if } x_{\text{min}}^{(j+1)} < x_n^{(j+1)} < x_{\text{max}}^{(j+1)} \\ \left(x_{\text{min}}^{(j+1)} + x_{\text{max}}^{(j+1)}\right)/2, & \text{otherwise} \ . \end{cases} \tag{6}$$

Here, bisection guarantees convergence, though bisection alone (without Newton iterations) would have much slower convergence. We pick our initial guess as the center of the interval, which places $x_r^{(0)}$ away from the two known critical points at the ends of the interval, where there is a greater likelihood of a failed Newton step.

Yet, bisection requires a finite interval. If either end of the interval is at $\pm\infty$, we start at an arbitrary distance δ away from the other end. If both ends are at $\pm\infty$, we start from an arbitrary point. Whenever Newton iteration fails, we simply move the guess by δ. When the interval becomes finite, we continue as explained above.

Another difficulty is the termination condition. The last step size of the Newton iteration does not bound the error for higher-order polynomials. Thus, we must continue until the interval is smaller than 2ϵ. On the other hand, Newton iterations often remain on one side of the root, so even when they converge to the exact solution (up to numerical precision), the interval can still remain arbitrarily large. To avoid this, when the last step size is below ϵ, we produce a guess that is likely to appear on the other side of the root. We achieve this by replacing the next guess in Equation 6 with

$$x_r^{(j+1)} = x_n^{(j+1)} + \begin{cases} \epsilon, & \text{if } \operatorname{sgn} f(x_{\text{min}}^{(j)}) = \operatorname{sgn} f(x_r^{(j)}) \\ -\epsilon, & \text{otherwise} \ . \end{cases} \tag{7}$$

If $x_r^{(j+1)}$ is on the other side of the root, we can return $x_n^{(j+1)}$ as our final guess. Otherwise, we end up with a new guess that is even closer to the root and we continue.

5 PERFORMANCE

The computation times of our method varies depending on the given polynomial. This is because we can quickly identify the absence of a root without any Newton iterations. This is used for early termination or skipping parts of the target interval. Also, the cost of numerical root finding depends on the desired accuracy.

In our tests with randomly-generated cubics, our method produced a similar average CPU performance as regular Newton iterations for cases when there is a root. Yet, when there are multiple roots, regular Newton iterations find one of them and not necessarily the one closest to the starting guess. More importantly, for about 22% of these random cubics, regular Newton iterations failed to find an existing root. Our method, on the other hand, has no such stability issues. In addition, for cases when there is no root, our method gets even faster by skipping numerical root finding. Regular Newton iterations, however, get even slower, resulting in up to 9× slower performance than our method in our tests (depending on the maximum iteration count used with regular Newton iterations).

The analytical solution for cubics performed about 2.5× slower than ours when there is a root and 6× slower when there is no root.

The cost of our method increases with increasing polynomial degree. Quartic (fourth order) polynomials took 2.6× and degree 10 polynomials took 23× longer than cubics on average in our tests.

As compared to other (real and complex) root finders, with cubics we observed that our solution is 12.9× faster than RPOLY [Jenkins and Traub 1970] and 107× faster than the QR factorization method in the Eigen library [Guennebaud et al. 2010] on average. With degree 10 polynomials our method maintains a performance advantage of 5.7× over RPOLY and 60× over the QR method.

6 CONCLUSION

We have presented a fast and robust algorithm for solving cubic and higher-order polynomials. For cubics with no real root within the interval, it incurs a minor additional cost over the analytical solution for quadratics. The cost for root finding depends on the desired accuracy. It is efficient for higher-order polynomials as well. An implementation of our method is available online [Yuksel 2022].

REFERENCES

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.
M. A. Jenkins and J. F. Traub. 1970. A Three-Stage Algorithm for Real Polynomials Using Quadratic Iteration. *SIAM J. Numer. Anal.* 7, 4 (1970), 545–566.
Cem Yuksel. 2022. Polynomial Roots. http://www.cemyuksel.com/?x=polynomials.