Acta Cybernetica **21** (2013) 191–203.

Geometric Newton-Raphson Methods for Plane Curves

Gábor Valasek*, Júlia Horváth, András Jámbori, and Levente Sallai

Abstract

Our paper reviews Kallay's results on a geometric version of the classic Newton-Raphson method, in the context of plane curve queries, e.g. curve-curve intersection, point-curve distance computation. Variants of the geometric Newton-Raphson methods are proposed and empirically verified. **Keywords:** curves, distance, intersection

1 Introduction

Plane curves are fundemantal tools in many applications, ranging from being building blocks of aesthetically pleasing figures [5][9], to defining NC machine tool paths [3] or highway roads [4].

Many applications require certain queries to be carried out on these curves, e.g. finding the closest point of the plane curve to a given point in the plane, or finding the intersection of two plane curves.

These queries can be formulated as systems of nonlinear equations, and to answer one of these queries, the corresponding system has to be solved. A popular choice for solving such systems of equations is the Newton-Raphson (NR) method.

The NR method starts from an initial guess and refines it iteratively, producing a sequence of guesses converging to the roots of the equations, provided an appropriate initial guess was used.

The NR method creates a linear approximation of the problem at each guess, and the next guess is the solution of this linear approximation.

Kallay proposed a geometric Newton-Raphson iteration in [1]. At each step, the curves in the query are substituted by higher order geometric approximants at the current guess, and then the query in question is solved on these and the guess refinement makes use of the solution on the geometric approximant.

Our goal was to empirically compare the traditional and the geometric NR method, and to propose variants of Kallay's method that retain its robustness and low iteration counts, at a reduced computational cost, if possible.

*E-mail: valasek@inf.elte.hu

DOI: 10.14232/actacyb.21.1.2013.14

The paper is organized as follows. Section 2 briefly compares the traditional and geometric NR algorithms and reviews Kallay's main results. Section 3 details our main contribution, the variants of Kallay's scheme by geometric approximant choice (parabola instead of circle) and construction (derivative-free geometric approximant creation). Section 4 shows the plane curve queries that served as the test problems for the comparison of the various NR methods and that the 3-point circle fitting variant is a viable, derivative-free modification of Kallay's geometric NR method in these problems. Section 5 summarizes the results.

2 Geometric Newton-Raphson methods

The Newton-Raphson (NR) method is a powerful tool for solving nonlinear equations. It defines an iterative process, which given a suitable starting point, converges to the roots of the equations. Each step of the iteration refines the current guess by using the solution of a linear approximation of the problem at the current guess.

The classic NR method is sensitive to the choice of the initial guess, that is, its convergence is not guaranteed for abritrary starting points. If the initial guess is within a certain neighborhood of an isolated guess, then the NR method produces a sequence of guesses that converges to the root quadratically. If the iteration is started from the neighborhood of a multiple root, then the rate of convergence is linear. [10]

The idea of using higher order approximations within the NR method, instead of a linear one, to gain higher convergence rate has been present in the literature [8]. Let the $(m+1)$th derivative of the function, whose root we are looking for, be bounded. Then the mth degree Taylor polynomial will be an mth order approximation, yielding a convergence rate of $m+1$ (for isolated roots) [1].

Let us consider the case of finding the zeros of an univariate equation.

In the generalized NR method, the solution of a degree m equation is required in the guess-refinement step, making the cases $m > 2$ less practical.

In addition to being computationally feasible, the case of $m = 2$ also has a straightforward geometric interpretation: at each step, an approximating parabola has to be intersected with the x-axis. This version of the Newton-Raphson method has a convergence rate of 3. Lee et. al. [6] and Park et. al. [7] proposed the use of osculating circles instead of parabolas, which are also second order approximations of the curve, however, the computational overhead, resulting from the curvature computations, made their approach less practical. [1]

It was Kallay who noted, that for geometric problems involving plane curves, using osculating circles has advantages when the osculating circles are used to approximate the curves in the query instead of approximating the equations formalizing the query on the curves.

In [1] Kallay proposed a geometric variant of the Newton-Raphson method, which utilizes a transformation that preserves the geometric domain of the original query, yet it retains the attractive quadratic convergence speed properties of the original NR method and improves on robustusness (at the expense of computational

cost, required due to the construction of geometric approximations).

Kallay's geometric NR method used geometric approximants to the original curves at each guess, and solved the queries on these. Then the results on the proxies were transformed back to the domain of the original curves to compute the new guess.

Let $I \subset \mathbb{R}$ be an open set and $\mathbf{p}(t) : I \to \mathbb{R}^2$ a smooth, regular parametric plane curve. Let us consider the following simple, general pseudo-code for the Newton-Raphson methods:

```
x = InitGuess( p(t) )

while (not IsDone( p(t), x ))
{
  x = NextStep( p(t), x )
}
```

InitGuess() creates the initial guess for the root of the function $f(t)$, $f(t)$ being the mathematical formulation of the geometric query on the plane curve $\mathbf{p}(t)$.

IsDone() is the termination condition of the NR iteration, which will be discussed in more detail in section 4.

NextStep() is the guess-refinement step. In the case of the classic NR method, it would return $x - \frac{f(x)}{f'(x)}$.

For geometric NR methods, NextStep() can be defined as follows:

```
NextStep( p(t), x )
{
  proxy gp = ProxyCreate( p(t), x )
  proxy_solution = ProxySolveQuery( gp, x )
  return x + ProxyReparam( p(t), x, gp, proxy_solution )
}
```

ProxyCreate() constructs a geometric proxy for the curve, which approximates $\mathbf{p}(t)$ at the current guess x. ProxySolveQuery() computes the result of the query on the geometric proxy and ProxyReparam() transforms the solution back into the original curve's domain.

3 Variants of the geometric Newton-Raphson methods

In this section we overview the line and circle proxy creation based on differential geometry, required for Kallay's geometric NR method.

To be able to provide an empirical comparison of the use of different geometric proxies, we propose a parabola proxy creation heuristic.

It must be noted that the curvature computation comes with an overhead which makes the geometric Newton methods more expensive to implement. To alleviate

this, we propose line and circle proxy creation heuristics, that do not require the evaluation of the derivatives of the curve.

3.1 Line proxies

Line proxies were used in Kallay's geometric Newton method when the curvature of the curve $\mathbf{p}(t)$ became 0 at the current guess x_n, x_n being a regular point of $\mathbf{p}(t)$. In this case, the tangent line was used as the proxy for $\mathbf{p}(t)$.

The line proxy can be represented by a point $\mathbf{p}_0 \in \mathbb{R}^2$ and a tangent direction vector $\mathbf{v}, |\mathbf{v}| = 1$. Its parametric form is $\mathbf{l}(t) = \mathbf{p}_0 + t\mathbf{v}$.

At the current guess x_n, $\mathbf{p}_0 = \mathbf{p}(x_n)$. We used the following two tangent direction definitions in the tests:

- Direction computed from the derivative of the curve: $\mathbf{v} = [\mathbf{p}'(x_n)]$, as Kallay
- Direction estimated from forward differences: $\mathbf{v} = [\mathbf{p}(x_n + h) - \mathbf{p}(x_n)]$, so that the derivaties do not have to be evaluated

where $[\mathbf{a}] = \frac{\mathbf{a}}{|\mathbf{a}|}$.

The second version comes from the geometric definition of the tangent line [2]: keeping $\mathbf{p}(x_n)$ fixed, the tangent line is the limit of the lines passing through $\mathbf{p}(x_n)$ and $\mathbf{p}(x_n + h)$ as $h \to 0$.

3.2 Circle proxies

The circle proxy can be represented by one of its points $\mathbf{p}_0 \in \mathbb{R}^2$, its normal $\mathbf{n}, |\mathbf{n}| = 1$ pointing towards the center of the circle, and its radius $r > 0$.

Kallay detailed the use of osculating circles as geometric proxies. The signed curvature of a smooth parametric curve can be computed as [3]

$$\kappa(x_n) = \frac{(\mathbf{p}'(x_n) \times \mathbf{p}''(x_n)) \cdot \mathbf{z}}{|\mathbf{p}'(x_n)|^3},$$

where $\mathbf{z} = \mathbf{x} \times \mathbf{y}$ denotes the unit vector perpendicular to the plane of the curve, $\mathbf{x}, \mathbf{y}$ being the orthonormal basis vectors of the plane of the curve.

If $\kappa(x_n) \neq 0$, the radius of the osculating circle of $\mathbf{p}(t)$ at x_n is $\rho(x_n) = \frac{1}{\kappa(x_n)}$. In this case, the circle proxy can be defined by setting $\mathbf{p}_0 = \mathbf{p}(x_n)$, $r = \rho(x_n)$, and $\mathbf{n}$ to the principal normal of the curve $\mathbf{p}(t)$ at x_n.

It is important to recall that the osculating circle of $\mathbf{p}(t)$ at x_n can be defined geometrically as well [2]: the osculating circle is the limit of the circles through $\mathbf{p}(x_n)$, $\mathbf{p}(y_k)$, $\mathbf{p}(z_k)$, where $y_k, z_k \to x_n$ as $k \to \infty$.

This leads to a circle proxy construction that does not require the computation of curvatures and derivatives:

Consider the circle through the points $\mathbf{p}(x_n - h)$, $\mathbf{p}(x_n)$, $\mathbf{p}(x_n + h)$, $h > 0$. Let us denote its center by $\mathbf{c}$ and its radius by R. Then the circle proxy can be defined by $\mathbf{p}_0 = \mathbf{p}(x_n)$, $\mathbf{n} = [\mathbf{c} - \mathbf{p}_0]$, $r = R$. Later we refer to the circle proxy constructed this way as the 3-point circle proxy.

3.3 Parabola proxies

The parabola proxy is represented by its apex $\mathbf{p}_0 \in \mathbb{R}^2$, an orthonormal frame $\mathbf{t}_0$ and $\mathbf{n}_0$, and $a \in \mathbb{R}$. The parametric form of the parabola we use is then

$$\mathbf{r}(t) = \mathbf{p}_0 + t\mathbf{t}_0 + at^2\mathbf{n}_0.$$

Recall that the curvature of $\mathbf{r}(t)$ at $t = 0$ is $\kappa(0) = 2a$.

An approximating parabola proxy to $\mathbf{p}(t)$ at x_n can be constructed by setting $\mathbf{p}_0 = \mathbf{p}(x_n)$, $\mathbf{t}_0 = [\mathbf{p}'(x_n)]$, $a = \kappa(x_n)/2$, and $\mathbf{n}_0$ to the principal normal of $\mathbf{p}(t)$ at x_n.

4 Testing

4.1 The problem

Let $I \subset \mathbb{R}$ be an open set and let us consider the problem of finding the closest point of a smooth, parametric plane curve $\mathbf{p}(t) : I \to \mathbb{R}^2$, to an arbitrary point of the plane, $\mathbf{q} \in \mathbb{R}^2$.

The above problem can be formulated as finding a curve parameter $t \in I$, such that it minimizes the squared distance

$$d^2(\mathbf{p}(t), \mathbf{q}) = \langle \mathbf{q} - \mathbf{p}(t), \mathbf{q} - \mathbf{p}(t) \rangle, \tag{1}$$

where $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the dot product of $\mathbf{a}, \mathbf{b} \in \mathbb{R}^2$.

Differentiating (1), we find that the parameter $t^* \in \mathbb{R}$ corresponding to the point of the curve that is closest to $\mathbf{q}$ should satisfy

$$\langle \mathbf{q} - \mathbf{p}(t^*), \mathbf{p}'(t^*) \rangle = 0,$$

where the prime denotes differentiation with respect to the curve parameter. Let $f(t) = \langle \mathbf{q} - \mathbf{p}(t), \mathbf{p}'(t) \rangle$. The classic Newton-Raphson method can be used to find the roots of $f(t)$.

Given an initial guess $x_0 \in \mathbb{R}$, let

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{\langle \mathbf{x} - \mathbf{p}(x_n), \mathbf{p}'(x_n) \rangle}{\langle \mathbf{x} - \mathbf{p}(x_n), \mathbf{p}''(x_n) \rangle - \langle \mathbf{p}'(x_n), \mathbf{p}'(x_n) \rangle}, \quad n = 1, 2, ... \end{aligned}$$

If x_0 is chosen from within a neighborhood of an isolated root, this yields a quadratically converging iteration. If x_0 resides in a neighborhood of multiple roots, then the rate of convergence is linear. It is important to note, however, that the iteration may not converge at all.

One can plot convergence figures of Newton-maps $N_p(z) = z - \frac{p(z)}{p'(z)}$ with various functions $p(z)$. E.g. Figure 1 shows which initial guesses of the complex plane result

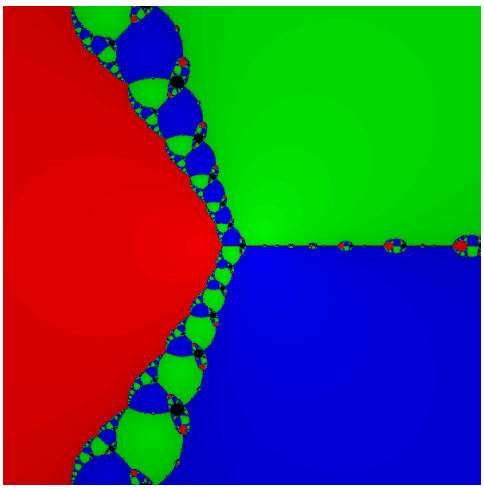


Figure 1: The application of the Newton-map $N_p(z) = z - \frac{p(z)}{p'(z)}$ to the polynomial $p(z) = z^3 - 2z + 2$, $z \in \mathbb{C}$. The red, green, and blue points of the complex plane denote initial guesses that create convergent iterations to different roots. The black points, forming Julia sets, denote elements of the complex plane, that do not yield a convergent Newton-Raphson iteration, if used as initial guesses.

in a convergent Newton-Raphson iteration in the case of $p(z) = z^3 - 2z + 2$, $z \in \mathbb{C}$. The discussion of starting point selection - which also affects the geometric NR methods - falls beyond the scope of our paper, we refer the interested reader to e.g. [10].

4.2 Test framework

The implementation of the NR methods was based on the pseudo-code of the generalized NR algorithm listed in section 2.

Each variant of the Newton-Raphson method uses the same InitGuess method, so that each algorithm starts from the same initial guess. We have taken equidistant parameter values and computed which one creates the closest point on the curve to $\mathbf{q}$ and used it as x_0.

The termination condition IsDone() has to be chosen carefully. Since the traditional NR solution finds a root of the function

$$f(t) = \langle \mathbf{q} - \mathbf{p}(t^*), \mathbf{p}'(t^*) \rangle,$$

the termination condition $|f(x)| \leq \epsilon$ seems to be a reasonable choice at first. However, upon closer inspection, one can find that this condition is very sensitive to the parametrisation of the curve. That is, if the magnitude of $\mathbf{p}'(t)$ over I is bounded and $M = max\{|\mathbf{p}'(t)| : t \in I\}$, $N = max\{|\mathbf{q} - \mathbf{p}(t)| : t \in I\}$, then the

reparametrisation $t \leftarrow \frac{\epsilon}{NM}t$ will make any initial guess $x_0 \in I$ the final guess as well, since

$$\begin{aligned}|\langle \mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM}), (\mathbf{p}(\frac{\epsilon t}{NM}))' \rangle| &= \frac{\epsilon}{NM}|\langle \mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM}), \mathbf{p}'(\frac{\epsilon t}{NM}) \rangle| \\ &\leq \frac{\epsilon}{NM}|\mathbf{q} - \mathbf{p}(\frac{\epsilon t}{NM})| \cdot |\mathbf{p}'(\frac{\epsilon t}{NM})| \\ &\leq \frac{\epsilon}{NM} NM = \epsilon.\end{aligned}$$

The iteration never start, since the termination condition is true for any initial guess.

Choosing $|x_{n+1} - x_n| < \epsilon$ for IsDone() instead, to anticipate the slow-down of root refinement, is still parametrisation-dependent, but to a much smaller extent than the previous one.

Given the fact that the domain is geometric in the case of geometric NR methods, one is motivated to find purely geometric termination conditions, which are also parametrisation independent. An example of this is the following:

Let the algorithm stop once the tangent at the current guess is (very close to being) perpendicular to the vector pointing from the current guess to $\mathbf{q}$. In practice, this can be checked more easily by comparing the cosine of the angle between the difference vector $\mathbf{q} - \mathbf{p}(t)$ and $\mathbf{p}'(t)$. Using this, the algorithm stops when $|\langle [\mathbf{q} - \mathbf{p}(t)], [\mathbf{p}'(t)] \rangle| < \epsilon$, where $[\mathbf{a}] := \frac{\mathbf{a}}{|\mathbf{a}|}, \mathbf{a} \neq \mathbf{0}$. Please note, that this is the geometric interpretation of condition $|f(t)| \leq \epsilon$.

The geometric, parametrisation-independent version of condition $|x_{n+1} - x_n| < \epsilon$ would be to stop the iteration if the arc-length between x_n and x_{n+1} is smaller than ϵ. In practice, however, it is more efficient to use $|\mathbf{p}(x_{n+1}) - \mathbf{p}(x_n)| < \epsilon$.

In addition to the above, IsDone should return true when the iteration count has exceeded a certain limit.

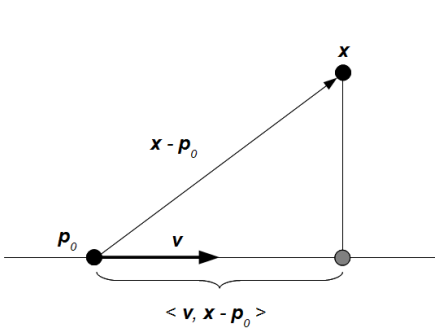
NextStep() is $x = x - \frac{f(x)}{f'(x)}$ in the case of the traditional NR method. The implementation of the geometric NextStep() is detailed in the following subsection.

4.3 Implementation of the geometric NR methods

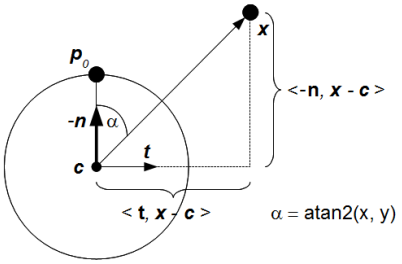
Section 3 has shown the proxy creation strategies for the various proxies, required for ProxyCreate(). In this subsection ProxySolveQuery() and ProxyReparam() are being investigated.

Finding the closest point of a line proxy to a given point in the plane is straightforward. Let ProxySolveQuery() return $t_n = \langle \mathbf{v}, \mathbf{q} - \mathbf{p}_0 \rangle$, as explained in Figure 2a. The closest point $\mathbf{x}$ of the line proxy to $\mathbf{q}$ is $\mathbf{x} = \mathbf{p}_0 + t_n \mathbf{v}$.

In general, the ProxyReparam() function uses the following strategy, as proposed by Kallay [1]: compute the arc-length s_n on the proxy between the current point of the iteration and the solution of the query. Let us estimate the parameter difference required to travel s_n along the original curve! This can be achieved by



(a) Finding the closest point $\mathbf{x}$ of a line, represented by one of its points $\mathbf{p}_0$ and its tangent direction $\mathbf{v}$, to a point $\mathbf{q}$ in the plane



(b) Finding the closest point $\mathbf{x}$ of a circle, represented by one of its points $\mathbf{p}_0$, the unit normal vector pointing from $\mathbf{p}_0$ to the center and the radius r, to a point $\mathbf{q}$ in the plane

Figure 2: Finding the closest point $\mathbf{x}$ of line and circle proxies to a given point $\mathbf{q}$ in the plane.

assuming that the parametric speed along the original curve can be estimated by $|\mathbf{p}'(x_n)|$ sufficiently, and setting $\Delta t = \frac{s_n}{|\mathbf{p}'(x_n)|}$.

In the case of line proxies, ProxyReparam() returns $\frac{t_n}{|\mathbf{p}'(x_n)|}$.

Finding the closest point of a circle to $\mathbf{q}$ requires only elementary geometry as well, see Figure 2b. $\mathbf{c}$ denotes the center of the circle and $\mathbf{t} = [\mathbf{p}'(x_n)]$. The closest point of the circle to $\mathbf{q}$ is the closest intersection of the circle and the line going through $\mathbf{c}$ and $\mathbf{q}$. Then $\alpha = atan2(\langle \mathbf{t}, \mathbf{q} - \mathbf{c} \rangle, \langle -\mathbf{n}, \mathbf{q} - \mathbf{c} \rangle)$. ProxyReparam returns $\frac{r\alpha}{|\mathbf{p}'(x_n)|}$.

There is no elementary solution for this query in the case of parabola proxies. Let (x_0, y_0) be the coordinates of the query point in the coordinate system of the parabola (with origin $\mathbf{p}_0$, and $\mathbf{t}_0$, $\mathbf{n}_0$ as the x and y axes, respectively), $\mathbf{p}(x_n) = (x, y)$. Then

$$d^2(\mathbf{q}, \mathbf{p}(x_n)) = (x - x_0)^2 + (2ax^2 - y_0)^2,$$

from which it follows that

$$d^{2'}(\mathbf{q}, \mathbf{p}(x_n)) = 4a^2x^3 + (2 - 4ay_0)x - 2x_0 = 0$$

has to be solved using the formula for cubic equations.

Since the apex of the parabola is $\mathbf{p}(x_n)$, ProxyReparam returns

$$\frac{2ax\sqrt{4a^2x^2 + 1} + asin(2ax)}{4a|\mathbf{p}'(\mathbf{x_n})|}$$

4.4 Results

The tests consisted of generating 100 random Bezier curves, with random degree between 10 and 100, and 100 random points for each curve. Then the following NR algorithms were used to find the closest point of the curve to the given random point:

- The classic NR method
- Kallay's geometric NR method using osculating circle proxies
- A geometric NR method using tangent line proxies
- A geometric NR method using estimated tangent lines from forward differences
- A geometric NR method using estimated osculating circles (the circle through 3 points of the curve)
- A geometric NR method using osculating parabola proxies

Each method was called three times for every point to investigate the effect of the following iteration termination conditions:

- Condition 0: $|x_{n+1} - x_n| < \epsilon$
- Condition 1: $|\mathbf{p}(x_{n+1}) - \mathbf{p}(x_n)| < \epsilon$
- Condition 2: $|\langle [\mathbf{p}'(x_n)], [\mathbf{q} - \mathbf{p}(x_n]) \rangle| < \epsilon$

Everything else was the same for all methods, including initial guesses, comparison and error thresholds.

For each method, we collected the following data: the ratio the method yielded a convergent iteration within 100 steps, the average amount of steps until a root is found in the case of convergent iterations, the standard deviation of the average amount of steps (for convergent cases), and the success rate (how many times did the given method found the closest point on the curve to q compared to the other methods).

The focus of our interest in the curve-point distance tests was the average amount of steps required to finish the iteration, and to find the geometric NR variants, that can match the average iteration counts of Kallay's geometric NR.

The actual performance time depends on evaluation costs of the curve's points and derivatives, making it dependent on the problem and type of curves as well. In the case of integral polynomial curves, the classic Newton method performed by far the fastest - the proxy set-up and query evaluation costs surprassed that of the derivatives'. The classic NR was followed by the tangent line and 3-point circle proxy variants, then Kallay's. The osculating parabola proxy variant was the slowest.

Method	Eval. $\mathbf{p}(t)$	Eval. $\mathbf{p}'(t)$	Eval. $\mathbf{p}''(t)$
Classic NR	1	1	1
Kallay gNR	1	1	1
Tangent line gNR	1	1	0
Fwd diff line gNR	2	0	0
3-point circle gNR	3	0	0
Osculating parabola gNR	1	1	1

Table 1: The table shows at how many distinct parameter values the curve and its derivatives have to be evaluated during a single step of the given NR variants.

The derivative-free variants become more appealing when the evaluation of the derivatives become more expensive, e.g. in the case of rational curves. Table 1 shows the curve evaluation costs of the various methods.

The rate of convergence is an indicator of how sensitive the given variant is to the choice of the initial guess. A variant performing poorly in this regard might be better handled by a different initial guess strategy, however, that investigation is beyond the scope of our paper.

The win rate is the least decisive attribute in our the tests, it simply tells that among the algorithms that finished within the prescribed relative error, which one ceased its iteration at a point of the curve that is closest to the point in the query.

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	8.616472	3.61186	72%	15%
Kallay gNR	8.5057535	5.71219	73%	15%
Tangent line gNR	9.570799	4.6349697	56%	6%
Fwd diff line gNR	8.679719	5.4383	75%	23%
3-point circle gNR	3.8460969	2.1435	95%	37%
Osculating parabola gNR	8.092985	5.59519	68%	4%

Table 2: Test results using termination condition 0, relative error 10^{-6}.

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	14.52	6.06115	63%	41%
Kallay gNR	4.63	3.44918	88%	14%
Tangent line gNR	12.1	6.1206	53%	22%
Fwd diff line gNR	13.22	6.0626	44%	17%
3-point circle gNR	4	2.74608	79%	3%
Osculating parabola gNR	10.89	5.1206	59%	3%

Table 3: Test results using termination condition 1, relative error 10^{-6}.

Tables 3 and 4 show that the parametrisation-independent geometric termination conditions 1 and 2 put Kallay's and the 3-point circle fitting geometric NR

Method	Avg. iter. cnt.	Std. dev.	Converges	Wins
Classic NR	10.799472	6.65479	49%	37%
Kallay gNR	2.290768	1.47857	88%	11%
Tangent line gNR	12.099338	7.63352	35%	27%
Fwd diff line gNR	8.105676	7.21142	76%	1%
3-point circle gNR	2.8278252	2.25557	94%	5%
Osculating parabola gNR	6.3981366	7.74148	32%	19%

Table 4: Test results using termination condition 2, relative error 10^{-6}.

methods forward considerably in terms of average iteration counts. They also show that the 3-point circle fitting NR variant has the most similar characteristics to Kallay's geometric NR method in regards of average iteration count behaviour.

To evaluate its relative performance more in detail, table 5 shows the result of a more comprehensive test for the classic, Kallay's geometric, and the proposed 3-point circle fitting geometric NR methods, with termination condition 1. Termination condition 1 was chosen because it is the more general parametrisation independent termination condition.

A set of 30000 curves, chosen randomly from families of regular and non-regular curves, were used, and for each curve, 10 random points were generated to compute the closest point of the given curve to the random point. The curves used in the test were integral and polynomial Bézier curves, trigonometric curves, piecewise curves with first and higher order derivative discontinuities.

Method	Avg. iter. cnt.	Std. dev.	Converges	Total time
Classic NR	14.32251	6.06115	59%	173s
Kallay gNR	8.85953	5.64918	67%	470s
3-point circle gNR	9.20167	6.74608	75%	284s

Table 5: Test results using termination condition 1 on 30000 random curves, relative error 10^{-6}. Column Total time shows the run time of a given NR variant on the random curve and point set, in seconds.

The average iteration counts of the methods are closer to each other than in the case of integral polynomial curves. Kallay's geometric NR method still requires the fewest steps on average, but due to the computational cost of a single step, its run time is the longest. The classic Newton method finished the fastest, followed by the proposed 3-point circle NR variant.

Table 5 shows that the proposed 3-point modification of Kallay's method succeeded in speeding up the algorithm while retaining its robustness in the point-curve closest point computation problem.

4.5 Curve-curve intersection

An additional test was carried out for the comparison of the classic, Kallay's geometric, and the three-point circle proxy geometric Newton-Raphson methods, by computing the intersection of two plane curves, $\mathbf{p}(t), \mathbf{q}(s)$, i.e. solving $\mathbf{p}(t) - \mathbf{q}(s) = \mathbf{0}$ for t and s.

Tables 6 and 7 show the results of the intersection computation of 10000 random, integral Bézier curve pairs of random degree between 10 and 100.

As in the case of the curve-point closest point problem, the 3-point circle variant shows similar average iteration counts and standard deviations as Kallay's geometric NR method, at a reduced execution time. Due to the relatively higher complexity of the problem, and the elevated costs of derivative evaluations, the classic NR method's execution times were matched by the 3-point method.

In this problem, the cost of setting up and using proxies is offset by the fact that the 3-point method does not require derivatives.

Method	Avg.iter.cnt.	Std.dev.	Converg.	Wins	Total time
Classic NR	8.02	5.7	82%	47%	1009 ms
Kallay gNR	5.13	1.88	82%	19%	1376 ms
3-point circle gNR	6.72	3.59	85%	34%	1062 ms

Table 6: Curve-curve intersection, relative error 10^{-7}, terminating when the new guess is close to the pervious one. Column Total time is in milliseconds, showing the total amount of time required to compute the intersection of 10000 curve pairs.

Method	Avg.iter.cnt.	Std.dev.	Converg.	Wins	Total time
Classic NR	7.86	6.01	81%	47%	1381 ms
Kallay gNR	6.09	2.66	82%	18%	1552 ms
3-point circle gNR	6.74	3.39	89%	35%	1263 ms

Table 7: Curve-curve intersection, relative error 10^{-7}, terminating when the point of the new guess on the curve is close to the previous one. Column Total time is in milliseconds, showing the total amount of time required to compute the intersection of 10000 curve pairs.

5 Summary

Our paper briefly reviewed the geometric Newton-Raphson methods.

We proposed geometric proxy constructions that do not require the evaluation of curve derivatives, in order to lessen the computational cost of the original geometric Newton-Raphson method.

We have shown empirically, that our proposed 3-point circle fitting, derivative-free modification performs similarly to the original geometric Newton-Raphson

method at a reduced computational cost.

Future work includes the investigation of different reparametrisation strategies and their affect on the algorithms performance, and further geometric proxy constructions.

6 Acknowledgement

Research is supported by the European Union and co-financed by the European Social Fund (grant agreement no. TÁMOP 4.2.1/B-09/1/KMR-2010-0003).

References

[1] M. Kallay: A geometric Newton-Raphson strategy. Computer Aided Geometric Design 18(8): 797-803 (2001)

[2] M. P. Do Carmo: Differential Geometry of Curves and Surfaces. Prentice Hall, 1st edition, 1976.

[3] R. Farouki: Pythagorean-Hodograph Curves. Springer. 2008.

[4] D. J. Walton, D. S. Meek: G^2 curve design with a pair of Pythagorean Hodograph quintic spiral segments, Computer Aided Geometric Design, Volume 24 Issue 5, July, 2007, Pages 267-285

[5] Ling Xu and David Mould. Magnetic Curves: Curvature-Controlled Aesthetic Curves Using Magnetic Fields. pages 18, Victoria, British Columbia, Canada, 2009. Eurographics Association.

[6] Lee, I.-W., Jung, G.-H., 1995. A generalized NewtonRaphson method using curvature. Volume 11, Issue 9, pages 757763, September 1995

[7] Park, B.-K.,Hitotoumatu, S., 1988. A new method using the circles of curvature for solving equations in R1. Publ. RIMS, Kyoto University 14, 249252.

[8] Phillips, G.M., Taylor, P.J., 1973. Theory and Applications of Numerical Analysis. Academic Press.

[9] G. Klár, G. Valasek: Employing Pythagorean Hodograph Curves for Artistic Patterns. Acta Cybernetica, Volume 20 Issue 1, January 2011, Pages 101-110

[10] Acton, F.S., 1990. Numerical Methods That Usually Work. Mathematical Association of America.

[11] John Hubbard, Dierk Schleicher, Scott Sutherland. How to Really Find Roots of Polynomials by Newton's Method, 1998