UDC 004.4'22 *doi: 10.30970/vam.2023.31.11974*

ONE APPROACH TO DRAWING IMPLICIT 2D CURVES USING INTERVAL ANALYSIS AND RUNGE METHOD TYPE

M. Oleksyn, P. Venherskyi, Y. Kokovska

Ivan Franko National University of Lviv,
1, Universytetska str., 79000, Lviv, Ukraine
e-mail: mykhailo.oleksyn@lnu.edu.ua, petro.venherskyy@lnu.edu.ua,
yaryna.kokovska@lnu.edu.ua

Implicit curve approximation stands as a foundational task in both mathematics and computer graphics. Numerous approaches have been introduced to tackle this problem, from algebraic techniques to numerical methods. Among these, interval analysis has gained significant attention. Its strength lies in its inherent ability to provide guaranteed bounds, ensuring a robust estimate of the curve's location, a feature not consistently offered by many other methods. In this paper, we introduce an enhancement to the curve approximation algorithm, building upon the established foundation set by the interval algorithm that takes into account global parametrizibility provided by John Snyder. Our primary objective is to optimize efficiency, with a particular emphasis on improving the boundary intersection determination. Our ambition is to speed up implicit curve approximation by using Runge method type instead of more traditional approach of using Newton method to take the advantage of derivative calculation complexity.

Key words: interval analysis, implicit curves, computer graphics, curves approximation

1. Introduction

Curve approximation is a key topic in computational geometry. It's crucial for tasks ranging from creating visuals in computer graphics to scientific visualizations. One of the methods that stands out for curve approximation is interval analysis. Instead of working with just points, interval analysis looks at a range of values. This can lead to better accuracy because it considers more information about a function over a specific range. However, while it can give us more precise results, it can also be computationally intensive.

At the core of using interval analysis for curve approximation is finding where the curve meets the edges of our chosen interval, or "boundary intersection detection". Over time, many techniques like the Newton method and binary search have been used for this purpose. These methods are popular because they're straightforward and generally work well. There are also more advanced methods, like the Hansen-Greenberg, that look more closely at how a curve behaves within an interval. However, each method has its challenges, from taking too much computation time to needing extra details about the curve.

This is where the Runge method type comes in. It's an approach mostly used for solving differential equations, but it has potential for curve approximation too. The main advantage of the Runge method type is its ability to give a good estimate of a function's behavior over an interval without needing to check every single value in that range. This could lead to faster results without losing accuracy.

In this paper, we'll dive into how we can use the Runge method type with interval analysis to approximate curves. We want to see if using the Runge method type can offer

a faster way to approximate curves without losing the benefits of accuracy. Our goal is to give a fresh look at curve approximation using interval analysis and offer new ways to tackle the challenges in the field.

2. Evolving into the third dimension

While 2D curve approximation holds its own set of challenges, expanding to 3D presents a frontier filled with novel complications. Surfaces, described by implicit functions, encapsulate a richer structure and provide deeper insights, especially in applications like fluid dynamics, architectural modeling, and scientific visualization.

Our interval-based algorithm, which has shown promise in 2D, can be evolved to address the complexities of 3D. The main advantage we have is the rigorous and systematic nature of interval arithmetic. When dealing with surfaces, the accuracy in determining the boundaries becomes even more crucial. Using interval arithmetic can provide tighter bounds and better guarantees on the correctness of the approximated surfaces. Moreover, the boundary intersections in 3D involve not just points, but curves. Our methodology can be extended to not only find these curve intersections efficiently but also to ensure they are connected correctly to form the implicit surface.

Neural networks can further augment this process. By training them on a variety of 3D surfaces, we can utilize their predictive capabilities to guide our algorithm's subdivisions, focusing computational efforts on the more complex regions of the surface and speeding up convergence in simpler areas.

3. Foundations of Interval Curve Approximation

This section is aimed to describe basic principles of each step of curve approximation. We will describe the use of intervals, subdivision process and, most importantly, criteria of subdivision stop. This section will also emphasis the process of resulted intervals set border intersections lookup. We will examine different approaches of this process, they advantages and disadvantages. Additionally, we will explain edge cases when intervals still need to be subdivided even if criteria is met to avoid ambiguous results. Finally, this section will explore how to form the approximated curve by connecting correct intersection points.

3.0.1. Global parametrizability

Current section dives into all aspects of global parametrizability, property that allows to focus on important areas of the grid for specific curve and reject not relevant. We will describe the criteria that dictate global parametrizability, explain how the intervals are determined to satisfy this criteria and will cover some edge cases that are not handled by the algorithm.

3.0.2. Parametrizability criteria

A curve is globally parametrizable if there exists a bijective mapping between its parameter domain and the curve itself. In simpler terms, for each point on the curve, there should be exactly one parameter value that maps to it. Similarly, for each parameter

value, there should be at most one point on the curve it represents. Considering the fact that our case is 2D curve let us refer to Appendix A of [1] for more practical formulation. Let $f : R^n \to R^n$ be continuously differentiable in an interval domain X. As Appendix A of [1] states, we are considering an r-dimentional manifold defined as the solution to a system of $n - r$ equations in n parameters ($r \in \{0, 1, ..., n-1\}$) :

$$\begin{aligned} &f_1(x_1, x_2, ..., x_n) = 0 \\ &. \\ &. \\ &. \\ &f_{n-r}(x_1, x_2, ..., x_n) = 0 \end{aligned}$$

Given a set of parameter indices, B $= \{k_1, k_2, ..., k_r\}$, and interval $X \in I^n$, the subinterval X over B is a set depending on r parameters $(y_1, y_2, ..., y_r)$, $y_i \in X_{k_i}$, defined by

$$\left\{ x \in X | \quad \begin{array}{c} x_i = y_i, if i = k_j \in B \\ x_i \in X_i, otherwise \end{array} \right\}$$

$\square J_{\{k_1,k_2,...,k_r\}}(X)$ is interval Jacobian submatrix, as an $(n-r) \times (n-r)$ interval matrix given by

$$\square J_{\{k_1,k_2,...,k_r\}}(X) \equiv \left[\square \frac{\partial F_i}{\partial x_j}(X) \right]_{j \notin \{k_1,k_2,...,k_r\}}$$

F is an interval extension of f. According to IntervalImplicit Function Theorem, the solution of equations $f_i(x) = 0$ is globally parametrizable in r parameters indexed by $\{k_1, k_2, ..., k_r\}$ over X if

$$det \square J_{\{k_1,k_2,...,k_r\}}(X) \neq 0$$

Talking strictly, we need to calculate both partial derivatives of the input function, for x and y parameters respectively. In case at least one of those derivatives contains 0 for any specific interval, this interval is guaranteed to be parameterizable in at least one of parameters.

3.0.3. Criteria satisfaction and corresponding actions

There are multiple approaches we may want to consider during the recursive subdivision of the initial interval:

1. Even subdivision. Ignoring curve behavior inside each specific interval we might divide the interval into 4 smaller even parts until all the resulted intervals satisfy the criteria. As a result we will receive the set of square shaped intervals that will contain approximate curve.
2. We might want to ignore global parametrizability and proceed with the subdivision into 4 even parts until all intervals reach some size threshold. As a result, we will receive a set of intervals each of the same size.

3. On the other hand, we might be interested in some insights provided by global parametrizability and adjust the subdivision. Each specific interval might be parametrized for single parameter but not for the other. At this point we might subdivide an interval into 2 parts (instead of 4) thus forming non-squared rectangles and improving the precision and efficiency.

We will consider a third approach that avoids not needed subdivision until threshold and achieves better precision with rectangles.

3.0.4. Edge cases

There are some exceptional cases of curve behavior inside the interval that are not allowed by an assumption:

1. The curve shouldn't loop inside an interval and cross its borders simultaneously.
2. The curve should not lie entirely inside an interval without looping.
3. The curve should not cross interval border once without looping.

For more detailed information and visual examples please refer to section 4.1 of [1].

3.1. Boundary intersection algorithms

This section will examine multiple algorithms and approaches that narrow down line and implicit curve intersection lookup problem. We will dive into basic theoretical design and expected complexity of each of them. Finally, this section will provide some comparisons between them.

3.1.1. Binary search

Theoretically binary search is powered by Intermediate Value Theorem. It compares signs of the function values for both inf and sup of an interval and subdivides an interval iteratively in two halves in case signs differ.

In each iteration, the size of the interval is halved which gives us the $O(\log n)$ time complexity. The algorithm only needs to store current interval thus the space complexity is $O(1)$. This algorithm doesn t require knowledge of function s derivative (unlike Newton s method) and is efficient with a logarithmic time complexity. Generally slower than methods like Newton s method, which use derivative information for faster convergence.

Pseudocode

```
FUNCTION intervalBinarySearch(f, initialInterval, tolerance, maxIterations):
    n = 0
    currentInterval = initialInterval

    WHILE n < maxIterations:
        midPoint = getMidPoint(currentInterval)

        fLower = f(currentInterval.lower)
        fUpper = f(currentInterval.upper)
        fMid = f(midPoint)

        IFfLower * fUpper < 0:
          IFabs(fMid) < tolerance OR getWidth(currentInterval) < tolerance:
```

```
            RETURN midPoint
          ELSE:
            IFfLower * fMid < 0:
              currentInterval.upper = midPoint
            ELSE:
              currentInterval.lower = midPoint
        ELSE:
          RETURN "The function does not change sign on the interval. No root found."
        n = n + 1
    RETURN "Method did not converge"

END FUNCTION

FUNCTION MIDPOINT(interval):
    RETURN (interval.lower + interval.upper) / 2

FUNCTION WIDTH(interval):
    RETURN interval.upper - interval.lower
```

3.1.2. Newton method

Capitalizes on the function's derivatives to estimate its roots. Starting with an initial guess, the function value and its derivative are used to update the estimate progressively. The process is reiterated until the root's estimate is within the desired tolerance or a predetermined number of iterations have been executed. The Newton iteration for a function $f(X)$ can be given as:

$$X_{n+1} = (x_n - \frac{f(x_n)}{f'(X_n)}) \cap X_n, x_n \in X_n$$

Where $f^{'}(X_n)$ is the derivative of f with respect to x evaluated at X_n. If the width of X_{n+1} is sufficiently close to zero or if X_{n+1} is very close to X_n then the method has converged and X_{n+1} can be taken as the approximation to the boundary intersection. The process is repeated with X_{n+1} as the new starting point in case convergence criteria are not met.

Pseudocode

```
FUNCTION IntervalNewtonMethod(f, df, initialInterval, tolerance, maxIterations):

    n = 0
    currentInterval = initialInterval

    WHILE n < maxIterations:
        midPoint = MIDPOINT(currentInterval)

        dfInterval = df(currentInterval)
        IF0 IN dfInterval:
          RETURN "Derivative interval contains zero. Method may not converge."
```

```
        yInterval = midPoint - f(midPoint) / dfInterval
        nextInterval = INTERSECT(currentInterval, yInterval)

        IF EMPTY(nextInterval):
          RETURN "Intervals do not overlap. No root found."

        IF WIDTH(nextInterval) < tolerance:
          RETURN nextInterval

        currentInterval = nextInterval
        n = n + 1

    RETURN "Method did not converge"

END FUNCTION

FUNCTION INTERSECT(interval1, interval2):
    newLower = MAX(interval1.lower, interval2.lower)
    newUpper = MIN(interval1.upper, interval2.upper)

    IF newLower <= newUpper:
        RETURN Interval(newLower, newUpper)
    ELSE:
        RETURN emptyInterval()

FUNCTION EMPTY(interval):
    RETURN interval.lower > interval.upper
```

3.1.3. Runge method type

The Runge method type is using similar approach for estimations of roots by using function s derivatives. The main difference is to replace the most expensive derivative computation part with derivative estimation. More detailed information can be found in [2].

The Runge iteration for a function $F(x)$ can be given as:

$$X_{n+1} = (x_n - \frac{f(x_n)}{\frac{1}{4}f'(x_n) + \frac{3}{4}f'(x_n + \frac{2}{3}(X_n - x_n))}) \cap X_n, x_n \in X_n$$

Pseudocode

```
FUNCTION IntervalRungeMethod(f, df, interval, tolerance, maxIterations):
    n = 0
    currentInterval = interval

    WHILE n < maxIterations:
        lowerBound = f(currentInterval.lower)
```

```
    upperBound = f(currentInterval.upper)

    midpoint = MIDPOINT(currentInterval)
    IF (lowerBound <= 0 AND upperBound >= 0)
       OR (lowerBound >= 0 AND upperBound <= 0):
       yInterval = midpoint - (1/4 * df(midpoint) + 3/4 * df(midpoint +
                   + 2/3 * (currentInterval - midpoint ))) ^-1 * f(midpoint)
       nextInterval = INTERSECT(currentInterval, yInterval)

       IF width(nextInterval) < tolerance:
          RETURN nextInterval
       currentInterval = nextInterval
    ELSE:
       RETURN "Root not found in interval"

    n = n + 1

  RETURN "Method did not converge"
END FUNCTION
```

3.1.4. Comparison

In the pursuit of efficient boundary intersection algorithms for curve approximation, three notable contenders arise. Starting with the Binary Search, its chief advantage lies in its logarithmic time complexity, ($O(logn)$), combined with its inherent reliability, granted the function is continuous. However, this assurance might come at the cost of requiring more iterations for optimal accuracy.

Advantages:

1. Predictability: The number of iterations required to achieve a specific tolerance is relatively predictable. Each iteration halves the search interval.
2. Reliability: Always converges if the function changes sign on the given interval.

Disadvantages:

1. Convergence Speed: Convergence can be slower than methods like Newton's, especially if the initial interval is wide and the tolerance is very small.
2. Assumption: Assumes that the function changes sign on the interval. It cannot find roots where the function doesn't change sign.

Computational timing is $O(log(n))$

Newton's Method, on the other hand, boasts potential quadratic convergence when positioned close to the root, often translating to swifter accuracy gains than its binary counterpart. Yet, its efficacy is intimately tied to the aptness of the initial guess. Additionally, Newton's approach demands the computation of the function's derivative, which might not always be accessible or straightforward.

Advantages:

1. Fast Convergence: For functions that satisfy certain conditions, the method typically converges super-linearly. It can be much faster than the binary search method for certain problems.
2. Direct Use of Derivative: The method utilizes the function's derivative, which can provide additional information about the function's behavior.

Disadvantages:

1. No Global Convergence Guarantee: Depending on the initial guess and the function's nature, it might diverge or converge to an unwanted root.
2. Requires Derivative: Needs the function's derivative, which isn't always easily available or computable.
3. Sensitive to Initial Guess: The choice of the initial guess can heavily influence the algorithm's success.

Computational timing is $O(1)$ – $O(log(n))$: The method can converge in constant time for specific problems, but it might also take logarithmic time, similar to binary search, or even more, depending on the function and initial guess.

The Runge Method type is a tool more commonly associated with differential equations. Because of one of its applicants in the denominator does not involve derivative calculation and basically may be replaced with the normal function calculation instead of interval extention, it converges faster than Newton method. Basic calculations for specific equation shows around 4000 iterations for Newton method and around 3600 iterations for Runge method type. This fact also leads to faster execution of the calculation application.

Advantages:

1. Higher Order Convergence: Designed to be a higher order method which, for certain functions, means fewer iterations than simpler methods.
2. Takes Advantage of Function Characteristics: It uses more than just the function value and its derivative, which can make it more accurate for certain problems.
3. Avoids the most costly computation of derivative by using lower order derivatives

Disadvantages:

1. Complexity: In some cases might be more costly than the other two methods

$O(1)$ – $O(n)$: Convergence can be faster than both Newton and binary search for certain problems due to its higher order nature. However, each iteration is more computationally intensive. It can take constant time for some problems but may also require linear time or more, depending on the function and the algorithm's specifics.

3.1.5. Unambiguity

At this step we already have a set of intervals each is globally parametrizable and contains the list of points where the curve intersects with its borders. Next step is to connect these intersection points with lines to form visual approximation. Before this last step we need to take into account the case when the curve is crossing the interval multiple times thus producing multiple intersection points but not all of them should be connected inside this specific interval.

Interval with 4 intersections on borders is displayed on Figure 1. There are 5 possible cases how the curve could behave inside the interval:

1. Crossing a to b inside, touching c outside.
2. Crossing b to c inside, touching a outside.
3. Crossing a to b to c.
4. Crossing a to c inside, touching b outside
5. Touching a, b and c outside

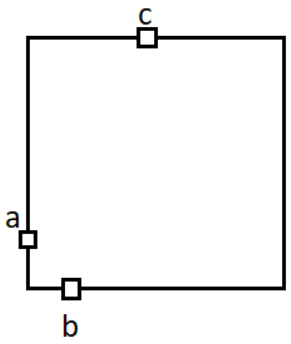

Fig. 1. Intersection example

We need to apply additional check to make sure how to connect these points inside an interval. To do this, we will first need to sort intersection points based on global parametrizability parameter. After this is done, we shall check if the curve connects two adjacent points by checking if the curve crosses the line that lies between them. Let us assume that an interval is parametrizable in x parameter and a, b, c and d on Figure 2 represent x coordinate. We now want to check if the curve connects points with coordinates b and c. For that purpose let us create line $y = d$,

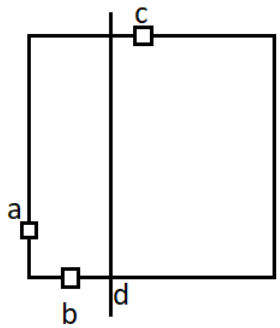

Fig. 2. Intersection verification

d is midpoint between b and c. We will apply the same method as we did for boundary intersection. In case the curve crosses the line, two points should be connected. We will repeat the same process for each pair of adjacent intersection points.

4. Algorithm

This section will describe how all previously explained parts are used as a part of an algorithm. Then we will provide a pseudocode. The algorithm is designed to approximate implicit curves within a given domain. Beginning with an initial interval and a specified inclusion function F the domain is iteratively subdivided. For each of the resulted intervals, the function's intersection points with the boundaries are determined. These intersections are then systematically sorted and connected to form an approximation of the curve, capturing its intrinsic characteristics within the specified domain.

4.1. Interval subdivision

The primary objective during this phase is to subdivide the given domain into intervals. The criterion for this subdivision is based on the global parameterization principle. The implicit curve contained in a proximate interval Y is called globally parameterizable in a parameter i if there is at most one point in Y on the curve for any value of the i-th parameter [1]. After this step a set of intervals are generated each is guaranteed to be parameterizable in one of the coordinates.

4.2. Boundary intersections

After identifying the intervals, the next step is to determine where the function intersects the boundaries of these intervals. This is crucial as it will guide the eventual approximation of the curve. Three methods were explored to perform this task: Binary search, Newton method and the Runge algorithm. After this step we will receive a set of boundary intersection points of each globally parametrizable interval.

4.3. Discarding ambiguous results

Once intersections on the interval boundaries are identified, the next step is to ensure they are disjoint. In case single interval contains multiple intersections on the opposite boundaries, we need to make sure that proximate intervals of intersections do not overlap. Otherwise the interval should be subdivided for further intersections recalculation.

When all intersections don t overlap, the next step is to sort these intersections. The sorting is based on the globally parameterizable argument. After sorting, algorithm checks if curve intersects the line that lies in between each two adjacent intersection points (e.g. if curve actually connects them). If so, we simply join two intersection point with the line, otherwise we proceed to the next pair of intersection points (if any).The connections are formed such that the curve's inherent properties and characteristics are retained to the maximum possible extent.

4.4. Algorithm steps

1. Initialize initial interval range, implicit function and calculate its partial derivatives.
2. If current interval globally parametrizable, got to step, otherwise split interval and start step 2 again for each of the subintervals.
3. For each interval boundary calculate intersections.
4. Sort intersections
5. In case interval contains more that 1 intersection points, check if the curve connects those points and discard points that are not connected to any other point.
6. Connect the rest adjacent intersection points with lines.

5. Conclusion

Efficient implicit curves approximation remains a crucial task. In this paper we discussed different approaches of boundary intersections lookup all based on [1] traditional algorithm. The key part was to replace well known Newton method with more efficient Runge method type [2] in order to avoid costly derivative calculation.

References

1. Snyder J.M. Interval analysis for computer graphics / J.M. Snyder //Computer Graphics. – 1992. – Vol. 26, (2). – P. 121–130. doi: 10.1145/142920.134024
2. Senio P.S. Solving systems of special form nonlinear equations by means of some modifications of Runge type interval iterative method / P.S. Senio, P.S. Vengersky // Interval Computations. – 1992. – Vol. 4, (6). – P. 59–65.
3. Lin L. Adaptive isotopic approximation of nonsingular curves / L. Lin, Ch. Yap. – 2009. – P. 351–360.
4. Burr M, Complete subdivision algorithms / M, Burr, Sung Woo Choi, B. Galehouse, Ch.K. Yap. – 2008. – P. 87-94.
5. Sampling and Meshing a Surface with Guaranteed Topology and Geometry / Siu-Wing Cheng, Tamal K. Dey, Edgar A. Ramos, Tathagata Ray. – 2007.

Article: received 29.09.2023
revised 25.10.2023
printing adoption 08.11.2023

ПІДХІД ДО ПОБУДОВИ НЕЯВНО ЗАДАНИХ 2D КРИВИХ, ВИКОРИСТОВУЮЧИ ІНТЕРВАЛЬНИЙ АНАЛІЗ І МЕТОДИ ТИПУ РУНГЕ

М. Олексин, П. Венгерський, Я. Коковська

Львівський національний університет імені Івана Франка,
вул. Університетська 1, Львів, 79000
e-mail: mykhailo.oleksyn@lnu.edu.ua, petro.venherskyy@lnu.edu.ua,
yaryna.kokovska@lnu.edu.ua

Наближення неявно заданих кривих є основним завданням у математиці та комп'ютерній графіці. Було введено численні методи для вирішення цієї проблеми, починаючи від алгебричних технік до чисельних методів. Серед них інтервальний аналіз здобув значну увагу. Його сила полягає у властивій здатності забезпечувати гарантовані межі, гарантуючи надійну оцінку положення кривої, функцію, яку не завжди надають багато інших методів. Ми пропонуємо поліпшення алгоритму наближення кривих, беручи за основу алгоритм Снайдера. Наша мета – оптимізувати ефективність, роблячи акцент на поліпшенні визначення перетину межі інтервалу. Ми прагнемо прискорити наближення неявно заданих кривих, використовуючи метод типу Рунге замість більш традиційного підходу за допомогою методу Ньютона.

Ключові слова: інтервальний аналіз, неявні криві, комп'ютерна графіка, апроксимація кривих.