CYBERNETICS and COMPUTER TECHNOLOGIES

UDC 519.85, 004.92 DOI:10.34229/2707-451X.25.1.2

O. FROLOV

EQUIPARTITION ALGORITHM FOR A FLAT PARAMETRIC CURVE BASED ON THE INTERSECTION BETWEEN IT AND A MOVING CIRCLE

The problem of equal-chord partitioning of a plane curve is applied from domains such as computer vision, robotics, signal processing, geographic information systems, and digital manufacturing. The proposed partitioning method is based on the intersection between a curve and a circle of constant radius centered on this curve, followed by moving the center to the intersection point. The designed algorithm consists of the following procedures: the procedure for the initial initialization of the radius of a circle based on a partition with a uniform distribution by a parameter, procedures for partitioning the curve by a circle for different directions of the circle`s move (direct, reverse, two-way); the procedure for obtaining an equal-chord partition with a specified tolerance of determining the chord length. For the real curve`s example, experiments were conducted on it equipartition by this algorithm, implemented in the Julia programming language, known for its high performance and ease of use in scientific computing.

Keywords: *pseudocode, iteration, computational complexity, segmentation, chord, intersection equation.*

Introduction. The problem of discretization of continuous geometric objects, a common issue in computational geometry, has broad interdisciplinary applications. From computer vision to robotics, signal processing, curve simplification in computer graphics applications, geographic information systems, and digital manufacturing applications, the need for the discretization and segmentation of plane curves is evident. These methods primarily aim to solve the problem of dividing the curve into 'homogeneous' segments with the same characteristics, such as equal length or curvature, or to minimize a predetermined error. This interdisciplinary approach underscores the broad applicability of the research and its potential impact across various fields.

The condition of partitioning the curve into points when the lengths of the chords connecting the segments are equal is an additional factor of practical interest. It allows, for example, to simplify the reproduction of a curve on CNC machines thanks to the constancy of the tool feed speed [1] or the reproduction of the movement of an object based on a video recording [2]. Therefore, the study of equal-chord segmentation methods, particularly their potential implementation in computer design and digital manufacturing systems, geoinformation systems, and computer vision systems, is of significant importance.

This work aims to develop new algorithms for partitioning flat parametric curves under the condition of equality of chords (chord length connecting segments of the partition) given the two outside points included in the first and last segment and given a number of segments. The research's novelty lies in applying an iterative procedure within the algorithm for partitioning a planar curve using a moving circle. The circle's radius is initialized based on a parameter-uniform curve segmentation, followed by radius adjustment to correct for uneven partitioning errors. With an increase in the number of segments, the proposed algorithms demonstrate linear complexity, which is much better than known solutions.

Related work. Numerous works are devoted to partitioning curves into segments, highlighting various methods, algorithms, and criteria for partitioning according to the specifics of the solved tasks. Among the significant works, it is possible to distinguish two sets: 1) works that examine curves already represented by a discrete sequence of points and 2) works that investigate the discretization of continuous curved lines represented in mathematical expressions.

Algorithms of adaptive partitioning of curves, represented by an ordered discrete set of points, make it possible to cut the original series of points by polyline segments with smaller nodes. Works [3–14] are devoted to this topic.

The algorithm presented in [3–5] is based on calculating the divergence, which is recognized by the maximum distance between the original curve and the broken one. At the initial stage, this method finds the point that is farthest from the segment connecting the initial and final points of the curve, compares the divergence with the specified error – ε , and in the case when this divergence is greater than ε, recursively calls itself on the sets of points from initial to the given and from the given to the final and so forth.

Algorithm [6] considers the partitioning condition in the form of the extremum of the ratio of the area of the curve bounded by the arc and the chord to the length of the chord.

In [7], an approach to choosing the number of points k located on the segmentation site was proposed. To find the correct value of k, they determined the best straight line for each k-point arc of the curve and calculated the root mean square error corresponding to that fit. This approach was developed in the work [8]. In [9], the method of polygonal approximation of a discrete curve based on the minimization of the integral root mean square error of approximation is considered.

In the study [10], an iterative approach was used to select the points of a discrete curve based on determining the admissible sector of the angles of a polyline segment starting at the current point of a given discrete set. In work [11], this algorithm was improved by optimizing it using the dynamic programming method.

An algorithm to determine dominant points by calculating the difference between the squares of the lengths of the curve's arc and the chord was considered in [12]. The article [13] proposed a heuristic approach to selecting initial dominant points with the subsequent insertion of additional dominants, provided that the required approximation accuracy was ensured.

In [14], an algorithm is considered that selects a subset of k from n points so that the difference in arc length between the approximation and the original curve is minimized. Given a limit of arc length divergence, the algorithm selects a subset of the minimum number of points necessary to bring the curve closer to this limit. No smaller subset of the starting points can reach this limit.

The works [15–22] are devoted to the algorithms for discretizing curved lines presented in a parametric or vector form.

In [15], the adaptive sampling algorithm of parametric curve approximation nodes is considered. At the same time, the following procedure is performed:

- The initial uniform selection of discretization nodes obtains the original ordered set of curve points.
- The output set is divided into intervals.
- For each interval, internal nodes are checked according to the chosen strategy for determining local flattening.
- Depending on the result of the check, the interval is either divided into two parts with a further recursive flattening check, or the extreme points of the interval are stored in a separate list of nodes that meet the flattening criterion.

The following criteria for checking local flattening are applied [15]:

- The area of the triangle formed by the two outside points and one inner point from the interval is relatively small.

- The angle formed by the outside left – P, inner – R, and outside right – Q points of the interval are obtuse and close to 180^0.
- The distance from the inner point R to the chord PQ, which contracts the outside points, is small.
- The expression $|\mathrm{P}-\mathrm{Q}|+|\mathrm{R}-\mathrm{Q}|$ is approximately equal to $|\mathrm{P}-\mathrm{Q}|$.
- Tangents to the curve at points P, R, and Q are approximately parallel.

The uniform sampling of the points from the parametric curve by the length of the arc is a fairly common problem from the point of view of practical applications. Such discretization in the case of polynomial curves requires numerical integration [16–18], which complicates implementation on specific devices. In [19], a method with an initial random sampling of curve points was considered to simplify the implementation.

The selection of curve points (in particular, NURBS) based on their reparameterization depending on curvature or mixed parameterization depending on arc length and curvature with equal weights was considered in [20]. A similar approach to the selection of curve points is given in [21], where the arc length and the bending energy of the curve (which depends on the square of the curvature) were taken as a mixed criterion. In [22], an asymptotically optimal approach is considered, where the number of sampling points is chosen depending on the distribution function similar to the curvature and is minimal for a given error of polygonal approximation of the curve. The mentioned methods [20–22] require numerical integration and solution of a system of nonlinear equations.

Papers [23–26] have focused on the problem of equal chord curve partitioning. The work [23] is theoretical and represents proof of the existence of an equipartition. The study [24] is more voluminous. In addition to theoretical material, an algorithm based on the piecewise linear approximation of the distance functions of two points of the curve was proposed. The work also analyzed the inequality errors of the partition's chords and the algorithm's computational complexity and presented the experimental results of the equipartitions. In the work [25] of the same authors, the abovementioned approach to partitioning under equality of distances was applied to polygonal approximation of curves in spaces of any dimensions under equality of approximation errors. In work [26], which concerns coating products of complex shapes on CNC machines by surfacing, an interpolation method called ECLD (equi-chord length deposition) was used. The ECLD algorithm uses the sequential determination of the curve parameter corresponding to the chord whose length differs from the specified one by a value that is less than or equal to the permissible error. In this case, partitioning with a fixed step and binary search is used to obtain the parameter's value. The example of the Rhinoceros 3D system of Robert McNeel & Associates should be given regarding the implementation of equal-chord partitioning in computer modeling and design systems. This system uses a curve-splitting tool – the divide command, which has options for equal chord lengths (EqualChordLength) [27].

The problem setting. Partitioning a parametric curve on the Euclidean plane into segments equal in chord length in the "classical" formulation [23, 24] was considered.

Let the equation of the curve be given in vector form

$$\mathbf{p}=\mathbf{p}(t). \tag{1}$$

It is necessary to determine the intermediate values $t_0<t_1<t_2<\ \ldots\ <t_{n-1}<t_n$ at the given interval of the curve parameter change $t[t_0,t_n]$ such that when substituting them into equation (1), we get the sequence of points – $\mathrm{P}\ \{\mathrm{P}_0,\ \mathrm{P}_1,\ \mathrm{P}_2,\ \ldots,\ \mathrm{P}_{n-1},\ \mathrm{P}_n\}$

$$\mathrm{P}_i=\mathbf{p}(t_i),\ (i=0,1,\ldots,n), \tag{2}$$

for which the condition of segment equality is satisfied:

$$d_1=d_2=\ \ldots\ =d_n, \tag{3}$$

where is the Euclidean distance from point P_0 to point P_1, etc.

Thus, a flat curve on the interval $[t_0, t_n]$ is divided into n equal chord length segments. The problem of the existence of such a partition was proved in [23, 24].

The proposed approach to solving the problem is based on the simple idea of dividing a flat curve by the imaginary movement of a constant radius circle along this curve. The circle successively intersects the curve from the side of the movement direction with the subsequent transfer of the center to the obtained point of intersection. If the center of the circle's initial position is placed at the start or end point of the curve, then to divide it into n parts, it is necessary to make n–1 such intersections.

To obtain an equal partition of the curve in this way, the following problems have to be solved:

1) determination of the circle radius so that it is equal to the length of the chord under condition (3);

2) point selection in cases where there is more than one point of a curve and a circle intersection in the corresponding direction.

The radius of the moving circle for a specific plane curve depends on the number of partitions. However, the second problem depends on the first problem's solution and the direction choice.

Fig. 1 illustrates the dependence of intersection points on the circle's radius. Comparatively small changes in the radius of the circle can give one (a circle with the intersection point $\mathrm{P}_{i+1}^{/}$), two (a circle with the points $\overline{\mathrm{P}}_{i+1}^{1}$ and $\overline{\mathrm{P}}_{i+1}^{2}$), three (circle with points P_{i+1}^{1}, P_{i+1}^{2}, P_{i+1}^{3}) and even more points intersection with the curve.

Fig. 2 makes it clear how, depending on the moving direction, the intersections that give the same common chord of the curve will determine different other points of intersection. So, the circle with the center at the point P_i gives three intersection points in a straight direction (fig. 2,*a*) P_{i+1}^{3}. If you move the center of the circle to the point $\mathrm{P}_j = \mathrm{P}_{i+1}^{3}$, then the circle of the same radius at the intersection with the curve in the opposite direction (fig. 2,*b*) will remain, except for the point P_{j-1}^{3}, which coincides with P_i, the points P_{j-1}^{1} and P_{j-1}^{2}, which differ from the points P_{i+1}^{1} and P_{i+1}^{2}.

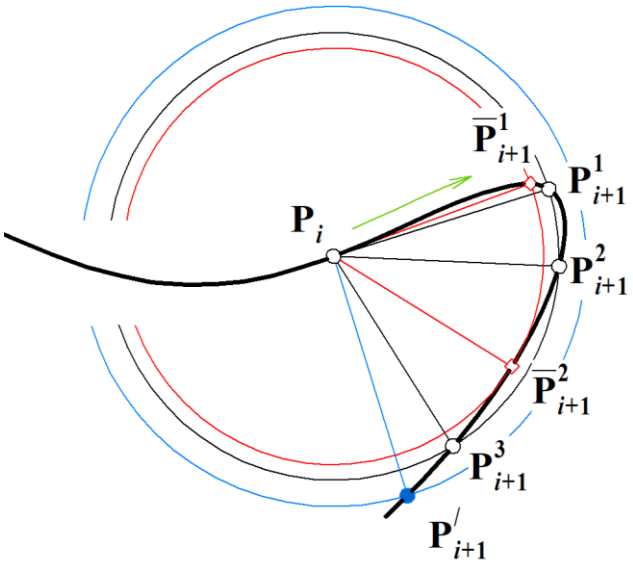


FIG. 1. Dependence of intersection points for the curve and the circle on the radius

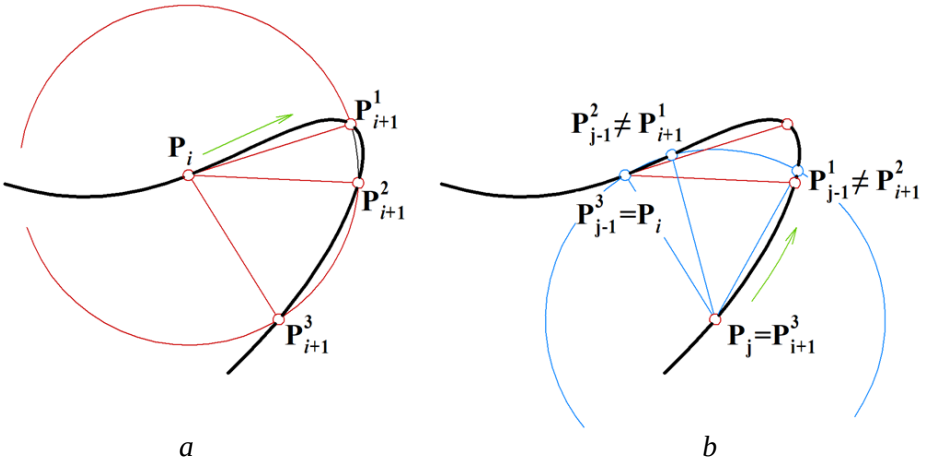


FIG. 2. Dependence of the intersection points location for the curve and the splitting circle on the moving direction

Since the presence of several intersection points significantly complicated the algorithm, within the framework of this study, the task was limited to solving the problem of determining the radius, assuming that there is only one point of intersection in the moving direction. In practice, this assumption is achieved starting from some value of n for a given interval of a specific curve. Therefore, the cases with existing multivalued intersection points are limited by n, which depends on the curve's properties and the segmentation interval's choice.

The proposed algorithm. The following approach is proposed for calculating the radius of a circle:

– In the first stage, the circle's radius is determined, ensuring segmentation with the intersection points' location on the curve inside the given interval. At the same time, we will have $n-1$ segments equal in chord length and a remainder, the chord length of which needs to be calculated;

– In the second stage, the value of the division radius is adjusted based on the difference between the received chord length of the remainder and the current radius of the circle. The splitting process is repeated with the new adjusted radius, and so on, until the difference between the chord of the remaining segment and the splitting radius is reduced to an acceptable error.

The algorithm presented in fig. 3 was used to implement the initialization procedure for the partition radius. This algorithm is based on the uniform parameter`s discretization for the given curve`s interval and calculating the lengths of all chords. The initial value of the circle's radius is taken as the statistical characteristic of the obtained chord sequence. Such a characteristic can be this sequence's minimum, average, or median value. The computational complexity of the initialization algorithm is $O(n)$.

After obtaining the initial value of the radius, it is necessary to split the curve directly by moving the circle. Since we already have two outside points of the curve, between which n-1 segmentation points need to be defined, we can move the circle in three ways:

– The partition begins from the circle's movement to the outside point corresponding to the value of the parameter t_0 – fig. 4,*a*. After making $n-1$ steps, we will get a partition and the remaining last segment between the points corresponding parameter`s values t_{n-1} and t_n. This version of the algorithm will be conventionally called the direct move;

– The partition begins from the circle's movement to the outside point corresponding to the parameter t_n value- fig. 4,*b*. After making the $n-1$ steps, we will get a partition with the remaining first segment (between the points corresponding to the parameter`s values t_0 and t_1). This version of the algorithm will be conventionally called the reverse move;

– The partition starts from outside points (t_0 and t_n), and two circles move toward each other – fig. 4,*c*. Let the circle move from the point corresponding to the parameter`s value t_0 and make the n_1 intersection steps. Then, the circle moving from the opposite direction must make $n-n_1-1$ intersection steps. The remaining segment will be located between the points corresponding to the parameter values t_{n_1} and t_{n_1+1}. This version of the algorithm will be conventionally called the two-way move.

INITIATE-RADIUS(t_0, t_n, n, *params*)

In: Starting point parameter value t_0, finish value of parameter t_n, n – number of segments in partition, *params* – list of curve's shape parameters.

Out: Initial circle radius r for evaluating partition.

Local: Sequence $\mathrm{P}\langle \mathrm{P}_0, \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n \rangle$ of $n+1$ points for uniform parameter step partition, sequence $D\langle d_1, d_2, \ldots, d_n \rangle$ of n segment length for uniform parameter's step partition.

1: $uniform-step \leftarrow (t_n - t_0)/n$ // calculate uniform step
2: **for** $i \leftarrow 0 \ldots n$ **do**
3: $P_i \leftarrow \mathbf{p}(t_0 + uniform-step \cdot i,\ params)$ // calculate curve's point for partition
4: **end for**
5: **for** $i \leftarrow 1 \ldots n$ **do**
6: $d_i \leftarrow \|\mathrm{P}_i - \mathrm{P}_{i-1}\|$ // calculate the length of i-th segment
7: **end for**
8: $r \leftarrow \tilde{d}$ // mean of D
9: **return** r

FIG. 3. Algorithm for initialization of the circle`s radius

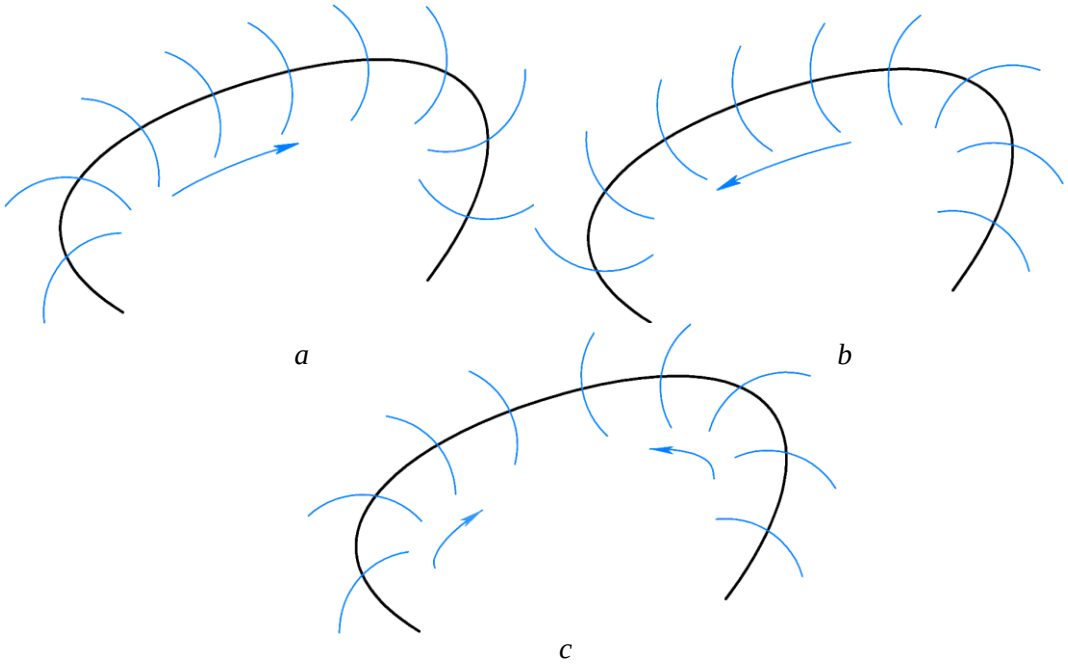


FIG. 4. Three methods of partitioning a curve by moving a circle: *a* – direct move; *b* – reverse move; *c* – two-way move

Fig. 5 presents an algorithm for direct and inverse partitioning of a flat curve by moving a circle. This algorithm sequentially calculates the coordinates of the intersection points based on the FIND-ROOT procedure. This procedure implements finding the intersection point between a circle and a curve, which depends on the curve's parametric form and the equation's solving method. The intersection equation in vector form has the following form:

$$\left\|\mathbf{p}(t)-\mathbf{p}_{ci}\right\|=r\,, \tag{4}$$

where $\mathbf{p}_{ci}$ is the current position of the circle's center, and r is the partition radius. From this equation, the FIND-ROOT procedure must determine its root, which is between the values of t_{ci} and t_n for the direct move (t_0 and t_{ci} – for reverse). The obtained parameter`s value is then used to calculate the intersection point coordinates according to the parametric equations and where the circle's center is moved on the next loop iteration. Since obtaining the intersection between a circle and a curve takes a constant amount of time, the computational complexity of this algorithm depends on the number of partitions, which is $O(n)$.

PARTITION1(t_0, t_n, n, r, *params*)

In: Starting point parameter value t_0, second value of parameter t_n for solver's initial conditions interval – $[t_0, t_n]$, n – number of segments in partition, circle radius r for evaluating partition, *params* – list of curve's shape parameters.

Out: Sequence P $\langle \mathrm{P}_0, \mathrm{P}_1, \mathrm{P}_2, \ldots, \mathrm{P}_n \rangle$ of $n+1$ partition points and corresponding sequence T $\langle t_0, t_1, t_2, \ldots, t_n \rangle$ of parameter's values.

Local: *start* – index of starting point, *stop* – endpoint index, *step* is 1 for direction from t_0 to t_n and –1 – otherwise.

```
if t0 < tn then // check direction to initiate parameters
    start ← 0
    stop ← n – 2
    step ← 1
else
    start ← n
    stop ← 2
    step ← –1
end if
P0 = p(t0 , params) // evaluate outside points
Pn = p(tn , params)
for i ← start…stop do
    ti+step ← FIND-ROOT( Pi , ti , params, r) // solve equation of an intersection circle and curve
    Pi+step ← p(ti+step , params) // evaluate inside points
end for
return P, T
```

FIG. 5. Pseudocode for the algorithm of partitioning a flat curve by direct or reverse move

Fig. 6 presents the algorithm for the two-way partition of a flat curve with a circle. Unlike the previous algorithm, this algorithm is adapted to receive only the partition's starting point and direction. To implement

a complete partition iteration, the algorithm must be called twice with the input parameter values corresponding to the left and right outside points and the number of segments n_1 and $n - n_1 - 1$. The order of the calls does not matter because they are independent.

PARTITION2(t_s, *n*, *r*, *direction*, *params*)

In: First point parameter value t_s, *n* – number of segments in partition, circle radius *r* for evaluating partition, *direction* is Boolean value (TRUE for direction from start point, FALSE – from endpoint), *params* – list of curve's shape parameters.

Out: Sequence P $\langle P_0, P_1, P_2, \ldots, P_n \rangle$ of $n+1$ partition points and corresponding sequence *T* $\langle t_0, t_1, t_2, \ldots, t_n \rangle$ of parameter's values.

Local: *start* – index of starting point, *stop* – endpoint index, *step* is 1 for direction from t_0 to t_n and –1 – otherwise.

```
1:  if direction = TRUE then // check direction to initiate parameters
2:     start ← 0
3:     stop ← n – 1
4:     step ← 1
5:     P_0 = p( t_0 , params) // evaluate outside point
6:  else
7:     start ← n
8:     stop ← 1
9:     step ← –1
10:    P_n = p( t_n , params) // evaluate outside point
11: end if
12: for i ← start…stop do
13:    t_{i+step} ← FIND-ROOT( P_i , t_i , params, r) // solve equation of an intersection circle and curve
14:    P_{i+step} ← p( t_{i+step} , params) // evaluate inside points
15: end for
16: return P, T
```

FIG. 6. Pseudocode for the algorithm of partitioning a flat curve by a two-way move

The complete procedure for equal chord partitioning a flat curve by a circle for direct or reverse move is presented in fig. 7. It is based on the previously described algorithms of the radius initial initialization (INITIATE-RADIUS) and partitioning the curve by a circle (PARTITION1). After assigning an initial radius value, this algorithm successively calculates the points of the curve partition by the circle in a while loop. After each partition, the remaining segment chord length is defined, and then the difference between it and the circle's radius is obtained.

Based on comparing the received error value with the permissible value of the partition unevenness, the algorithm decides to continue partitioning with the adjusted radius value or stop iterating with the received points of equal-chord partition. To change the radius, a uniform distribution of error between all segments was used:

$$r = \frac{\Delta}{n} + \bar{r}\,, \tag{5}$$

where Δ is the difference (error) between the chord of the residual segment and the radius of the partition circle and $\bar{r}$ is the "old" (previous) value of the radius; note that formula (5) gives the same result as

averaging the chord length over all segments. The computational complexity of the proposed algorithm depends on the number of partition points and is $O(k \cdot n)$. Here, k denotes the number of iterations required to achieve the given tolerance (evenness of partitioning).

EQUAL-PARTITION1(t_0, t_n, n, e, *params*)

In: Starting point parameter value t_0, finish value of parameter t_n, n – number of segments in partition, absolute tolerance e for difference between segments length, *params* – list of curve's shape parameters.

Out: sequence P $\langle P_0, P_1, P_2, \ldots, P_n \rangle$ of $n+1$ partition points and corresponding sequence T $\langle t_0, t_1, t_2, \ldots, t_n \rangle$ of parameter's values.

Local: Circle the radius r for evaluating the partition, the d_n – distance from (n-1)th to the nth point of the partition, and the *stop* – Boolean value for breaking iteration.

1: $r \leftarrow$ INITIATE-RADIUS(t_0, t_n, n, *params*) // initiate radius of partition
2: *stop* ← FALSE
3: **while** stop = FALSE **do**
4: $P, T \leftarrow$ EVAL-PARTITION1(t_0, t_n, n, r, *params*) // evaluating partition points
5: $d_n \leftarrow \|P_n - P_{n-1}\|$ // calculate the length of the remaining segment after partitioning
// $d_1 \leftarrow \|P_1 - P_0\|$ – for reverse direction
6: $sg \leftarrow \text{sgn}(t_n - t_{n-1})$ // sign parameters difference for remaining segment
// $sg \leftarrow \text{sgn}(t_1 - t_0)$ – for reverse direction
7: $\Delta = sg\ d_n - r$
// $\Delta = sg\ d_1 - r$ – for reverse direction
8: **if** $|\Delta| \le e$ and sg **then** // stop condition
9: *stop* ← TRUE
10: **else**
11: $r \leftarrow \Delta/n + r$ // new radius of partition
12: **end if**
13: **end while**
14: **return** P, T // points and parameters of equal partition

FIG. 7. Equal-chord partition algorithm for one-way move

Fig. 8 shows the complete procedure for equal-chord partitioning for the two-way move of a circle. This algorithm differs from the previous one in that the two calls mentioned above are used in the PARTITION2 procedure, which implements the partition of the curve from the start and end points. This results in two sequences of partition points starting from the outside points of the interval. The residual interval is between these two sequences' last points of intersection. When the desired partition tolerance is reached, the EVAL-PARTITION2 procedure returns the union of the specified sequences. We also note that since the calls to the PARTITION2 procedure are independent, they can be done in parallel to reduce the partitioning time. In this case, the computational complexity of this algorithm will be $O(k \cdot n/2)$.

Numerical experiments. Let's proceed to consider the experimental part of the work. The tasks of the experiments were to study the efficiency of the proposed algorithm and determine the influence of their parameters and options on the time of partition.

A program code was implemented in the Julia programming language. A double-precision format (float64) was used to represent the data. For the numerical solution of the equation, the package for finding the roots of nonlinear equations, NonlinearSolve, was used. The solve function, by default, implements a combined algorithm for finding roots by selecting a specific numerical method. It is also possible to set the method's permissible absolute or relative tolerance. The Threads and Distributed packages implemented the threads and processes computing models. Performance was measured using the BenchmarkTools package. The results were processed and visualized using MS Excel and the Plots package. All calculations were performed on a quad-core Intel Core i5-6300HQ processor laptop.

EQUAL-PARTITION2(t_0, t_n, n, n_1, e, *params*)

In: Starting point parameter value t_0, finish value of parameter t_n, n – number of segments in partition, n_1 – number of segments in direction from starting point, absolute tolerance e for difference between segments length, *params* – list of curve's shape parameters.

Out: Sequences $P' \left\langle P_0, P_1, P_2, \ldots, P_{n_1} \right\rangle$, $P'' \left\langle P_{n_1+1}, P_{n_1+2}, \ldots, P_n \right\rangle$ of n_1+1 points for the first partition and $n-n_1$ points – for the second partition, corresponding sequence $T' \left\langle t_0, t_1, t_2, \ldots, t_{n_1} \right\rangle$, $T'' \left\langle t_{n_1+1}, t_{n_1+2}, t_{n_1+3}, \ldots, t_n \right\rangle$ of parameter's values.

Local: Circle radius r is used for evaluating partition, d_b is the distance between the first partition's endpoint and the second partition's first point, and *stop* – Boolean value is used for breaking iteration.

1: $r \leftarrow$ INITIATE-RADIUS(t_0, t_n, n, *params*) // initiate radius of partitions
2: *stop* ← FALSE
3: $n_2 \leftarrow n - n_1 - 1$ // number of segments in direction from endpoint
4: **while** stop = FALSE **do**
5: P', $T' \leftarrow$ EVAL-PARTITION2(t_0, n_1, r, TRUE, *params*) // partitioning from starting point
6: P'', $T'' \leftarrow$ EVAL-PARTITION2(t_n, n_2, r, FALSE, *params*) // partitioning from end point
7: $d_b \leftarrow \left\| P_{n_1+1} - P_{n_1} \right\|$ // length of the remaining segment between partitions
8: $sg \leftarrow \text{sgn}(t_{n_1+1} - t_{n_1})$ // sign parameters difference for the remaining segment
$\Delta = sg\ d_b - r$
9: **if** $|\Delta| \le e$ and sg **then** // stop condition
10: *stop* ← TRUE
11: **else**
12: $r \leftarrow \Delta/n + r$ // new radius of partition
13: **end if**
14: **end while**
15: **return** $P' \cup P''$, $T' \cup T''$ // union of partitions points and parameters

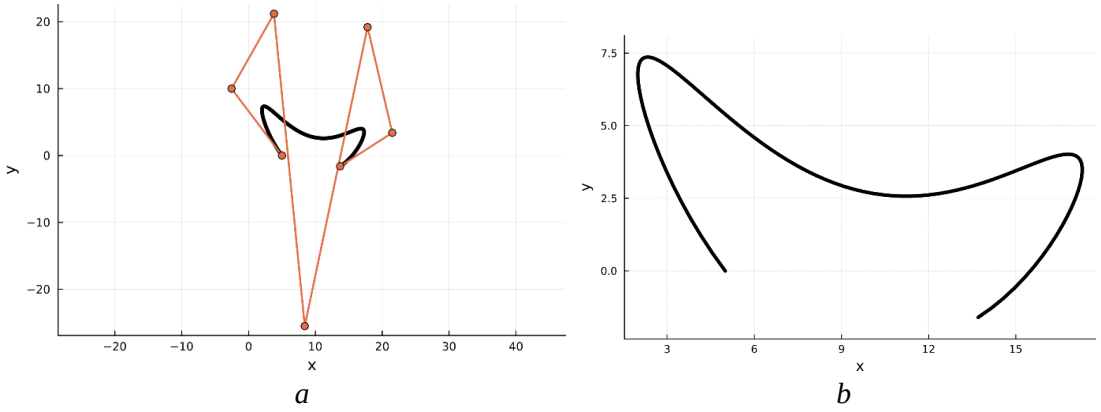
FIG. 8. Equal-chord partition algorithm for two-way move

A specific flat parametric curved line was chosen for segmentation – a Bezier curve of the sixth order inside a standard partition interval [0,1] when the outside points are the first and last points of its control polygon. The Bezier curve is presented in fig. 9.

The first part of the experiments aimed to study the value k of the number of iterations required for equal-chord partition of the selected flat curve with the given tolerance of defining the chord length. For

this purpose, the number of segments n was changed, and the following methods for determining the initial value of the radius were used – as the minimum value of the uniform parameter`s sampling, as its median, and as an average value. At the same time, all three variants of the proposed algorithm were applied. Experiments were carried out with chord inequality error values that depended on the number of segments according to the expression:

$$e = \frac{0.0001}{n}. \tag{6}$$



a *b*

FIG. 9. A flat Bezier curve for the experiments: *a* – the curve and its control polygon set by the points (5.0; 0.0), (–2.55; 10.0), (3.8; 21.2), (8.4; –25.5), (17.8; 19.2), (21.5; 3.4), (13.7; –1.6); *b* – the shape of the curve

The segments were divided in half between the sides for the two-way variant. That is, the value of n_1 was determined as:

$$n_1 = n \operatorname{div} 2. \tag{7}$$

It was found experimentally that the proposed algorithm can be applied for this curve starting with the number of partition segments $n = 27$. At the same time, since the values of k varied in a reasonably wide range, the graph was divided into two parts: from 27 to 100 and from 100 to 500 segments. Fig. 10 presents the number of iterations obtained depending on the number of partition segments.

The presented dependences for all algorithm modifications generally stabilize the value of k starting from $n \approx 45$–50, and after that, large fluctuations no longer occur. Stable values of $k < 10$ at $n \geq 100$ slowly decrease with increasing discretization.

The presence of "flashes" of k values on the graphs with a sufficiently small degree of partition ($n \leq 40$) can be associated with the complications of achieving the required uniformity at a particular n, which are caused by the shape of the curve and depend on the move direction when the algorithm becomes sensitive to small changes in the radius. This circumstance is confirmed by the "flashes" occurring at varying amplitudes for different directions and values of n. At the same time, for two-way moves, the maximum values of k were smaller, which can be explained by the presence of two moving points in the residual segment of the partition.

Regarding the influence of the statistic's choice for the initialization of the partition radius, the experiments' results showed that using the minimum chord value of the uniform by the parameter partition as the radius in all cases gave more iterations, which is reasonably expected. The results of the mean and median values were very close, but the mean value was more stable. Note that the indicated differences in the results were minor compared to n. Therefore, it was decided to conduct further experiments using the average value when initializing the radius.

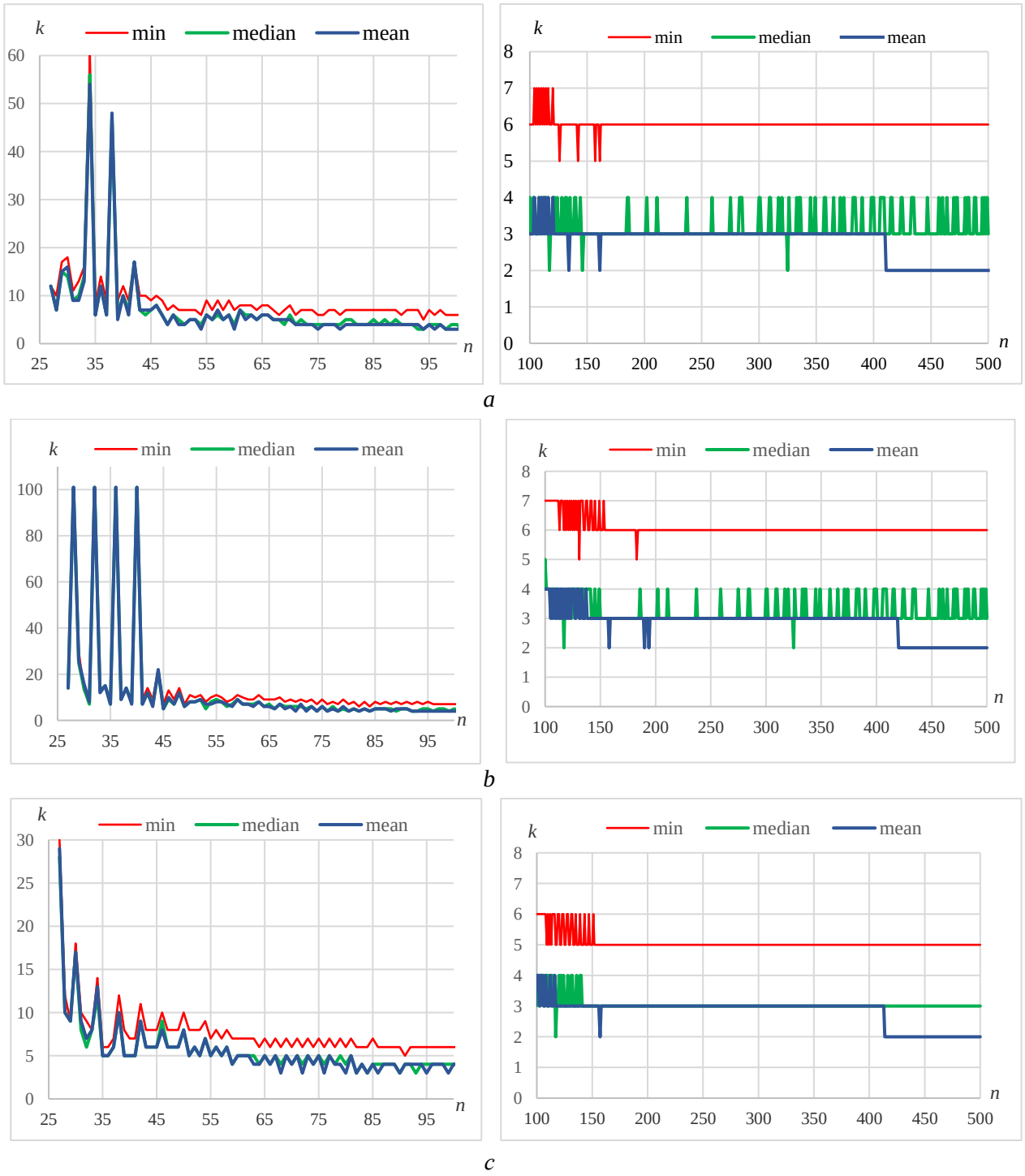


FIG. 10. Iterations depend on the number of partition segments with different statistics for obtaining the initial radius for three partitioning methods: *a* – direct move; *b* – reverse move; *c* – two-way move

The next stage of the experiments was devoted to studying the influence of the specified tolerance of determining the chord length on the number of iterations *k*. At the same time, similarly to the previous

experiment, the values of the number of iterations were obtained depending on the number of partition segments, but for different values of the tolerance *e*. The tolerances were represented by a discrete series of seven values from $1 \cdot 10^{-3}$ to $1 \cdot 10^{-10}$ incremented by 10^{-1}. Calculations were carried out for all three algorithm variants presented, with a degree of partition from 27 to 1000 segments. To visually explain the obtained results and reduce the volume of graphic material, it was decided to do the following – to take the ratio of the maximum number of iterations (for $e = 1 \cdot 10^{-10}$) to the minimum value of *k* (for $e = 1 \cdot 10^{-3}$), corresponding to the same value of *n*:

$$c = \frac{k_{max}}{k_{min}}. \tag{8}$$

Figure 11 presents graphs obtained from the results of the experiments. As we can see, all three algorithm variants needed to demonstrate a clear dependence of the ratio on the number of partition segments. The mean ratio values were almost the same for all variants. The increase in the number of iterations when the tolerance is changed turned out to be insignificant compared with the increase in accuracy (on average 2.5 times, while the accuracy increased by 10000000 times). For direct and reverse moves, it is possible to see the individual “flashes” in all *n*`s ranges on the graphs. For the two-way variant, "flashes" are present only in the initial section from 27 to 160 segments.

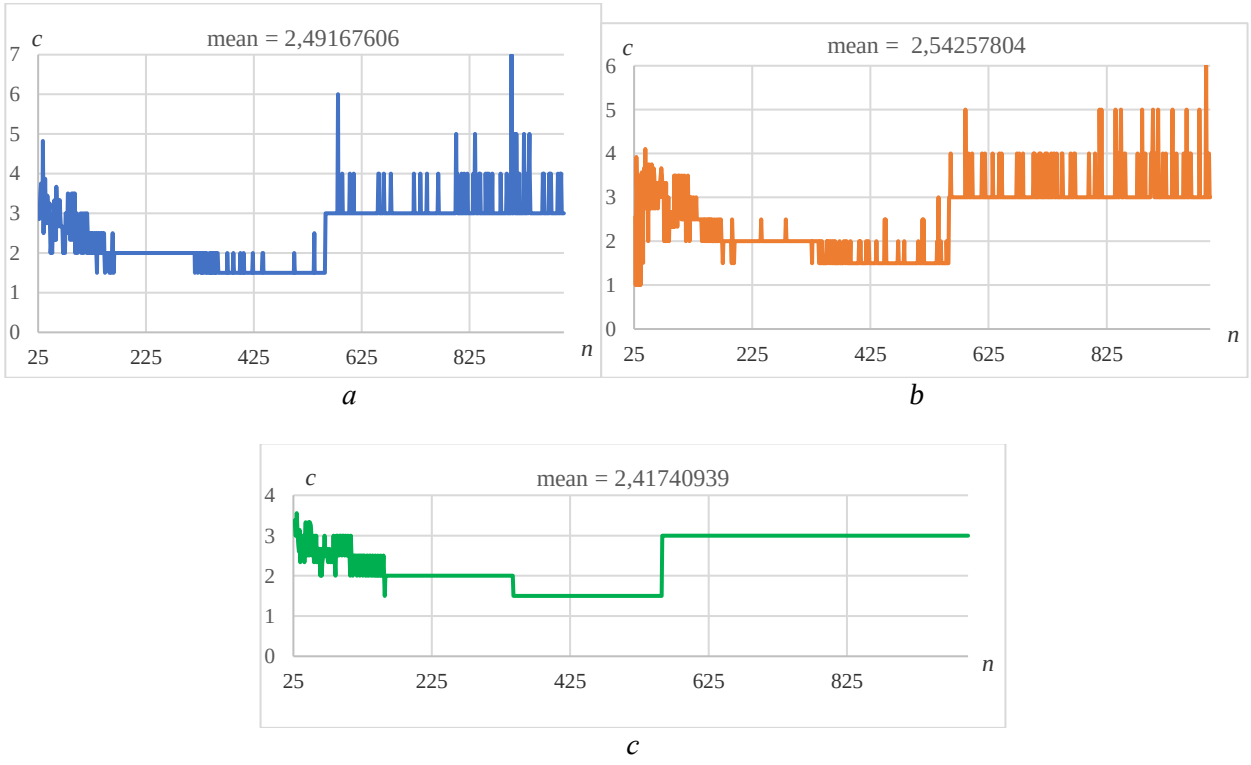
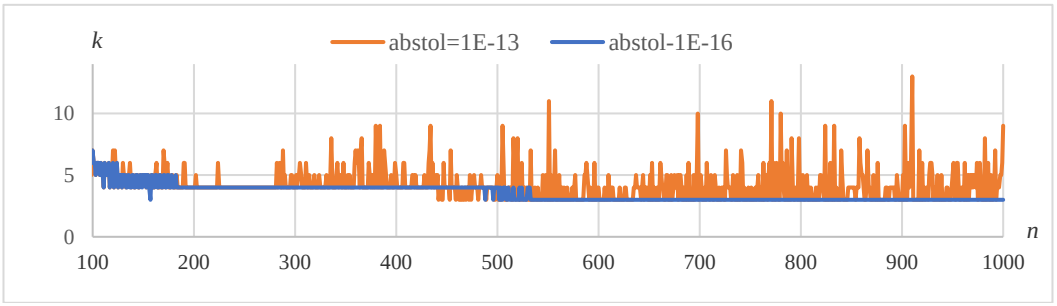


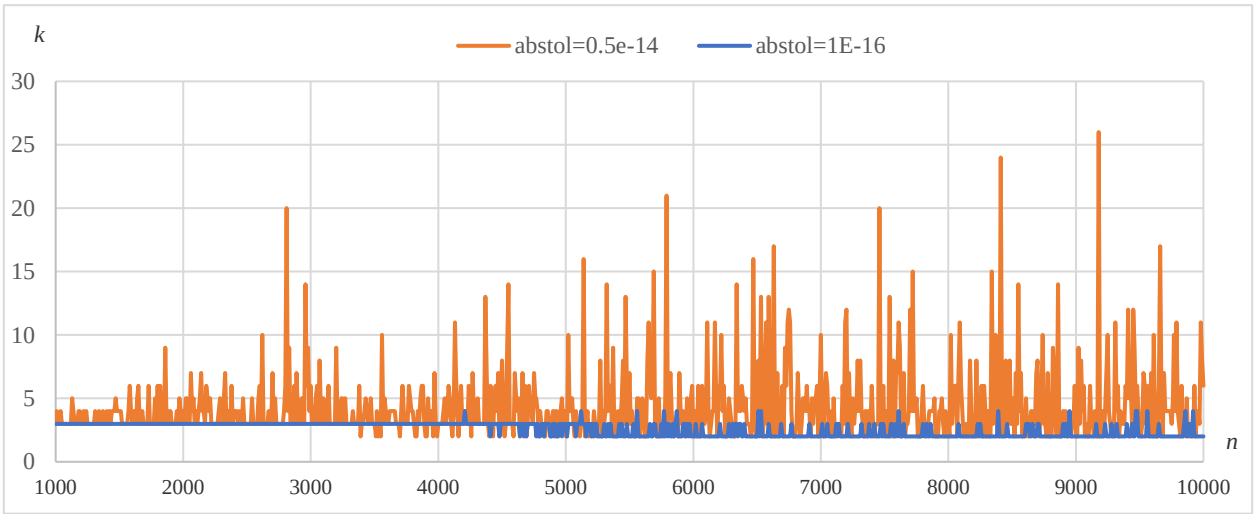
FIG. 11. Increasing the number of iterations when changing the partition radius tolerance from $1 \cdot 10^{-3}$ to $1 \cdot 10^{-10}$: *a* – direct move; *b* – reverse move; *c* – two-way move

With the simultaneous increase in the number of segments and tolerance value, the question of corresponding the tolerance value, the accuracy of the representation of numbers (the selected data type), and the tolerance of the intersection equation solve is raised. To study the influence of tolerance, the number of iterations was calculated at the $e = 1 \cdot 10^{-11}$ in the range of *n*`s values from 100 to 10000, and absolute tolerance of the numerical solve method was *abstol* $= 5 \cdot 10^{-13}$ and *abstol* $= 1 \cdot 10^{-16}$ for the two-way algorithm as

the most stable. The obtained results were divided into two parts by the values of n (fig. 12). This experiment demonstrated the increasing influence of the solving tolerance with the n's growth – for the less accurate method, the amplitude and frequency of k's "flashes" increased. Note that for the method tolerance value of $abstol = 1 \cdot 10^{-13}$, the algorithm gave unstable results and could not perform the partition for some values of n. Such results are explained by the fact that the method's accuracy did not allow for achieving the required deviation of the chord length when this floating-point format is used. In support of this assumption, changing the data type of the partition points from float64 to float128 eliminated the problem. After that, for the method tolerance value equal to $1 \cdot 10^{-13}$, the algorithm correctly partitioned the curve over the entire range of n`s values.



a



b

FIG. 12. Increasing the number of iterations for the partition radius tolerance $e = 1 \cdot 10^{-11}$ and different tolerance of the numerical solve method: *a* – $100 \leq n \leq 1000$; *b* – $1000 \leq n \leq 10000$

At the final stage, the execution time of the partition of the flat Bezier curve was measured at different n. These experiments aimed to compare different versions of parallel partitioning and its sequential implementation for the algorithm using a two-way move. With this approach, the value of k for a particular n remains unchanged, but thanks to the possibility of moving from both sides, the partition can be done in parallel in time. So, simultaneously with the sequential version, the following were considered:

1) three threaded versions of the algorithm:

a) two threads calculate the partition points from the shared memory (array), the reference to which is passed to the procedure;

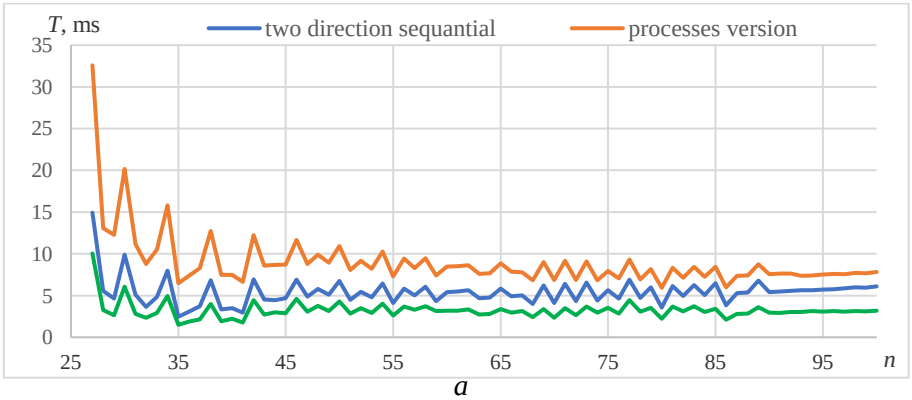
b) two threads calculate the partition points with their location in two arrays (each stream calculates its array); the arrays are transferred by reference;

c) two threads calculate the partition points with their location in two arrays (each stream calculates its array); the threads return the arrays;

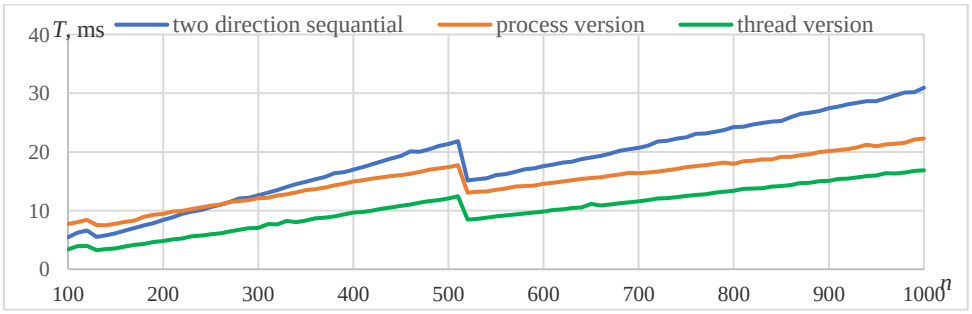
2) the algorithm's process version with the partition points' location in the shared memory of the two processes.

The measurement of the execution time was carried out with the same values of the radius tolerance $e = 1 \cdot 10^{-7}$ and the accuracy of the solving numerical method. Next, for each *n*, execution time statistics were obtained based on the results of a certain number of tests – the *samples* parameter. Statistics were minimum, maximum, mean, and median time.

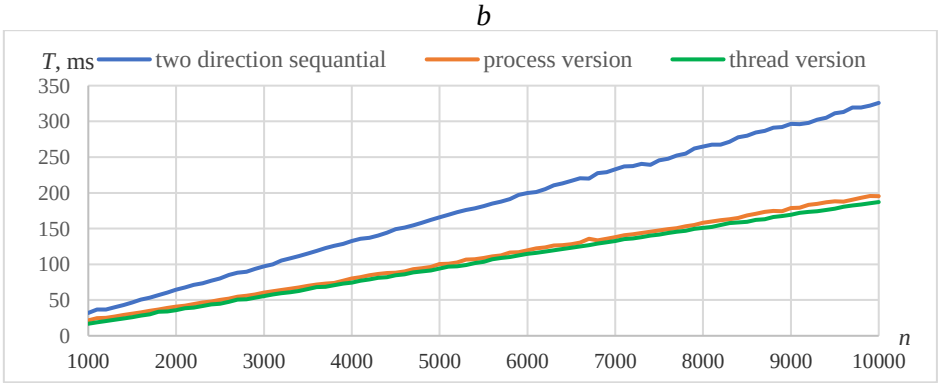
The measurement results were very close for all three threaded versions over the entire range of *n* values, so it was decided to show them as a single version on the graphs. Fig. 13 presents the experiment's results, where the median execution time is displayed depending on the number of partition segments. At the same time, the range of change of *n* from 27 to 10000 was divided into three parts.



a



b



c

FIG. 13. The median of equal chord partition execution time is dependent on the number of segments: *a* – 27 ≤ n ≤ 100, samples = 500; *b* – 100 ≤ n ≤ 1000, samples = 200; *c* – 1000 ≤ n ≤ 10000, samples = 100

Analyzing the obtained dependencies, we can note that for small *n*, it is impractical to perform the partition using two processes since the costs of organizing inter-process interaction do not allow to ensure a result better than the sequential version. Starting with $n = 270$, these costs are compensated by the parallel execution of the partition, and the process version begins to prevail over the serial version. The threaded version of the algorithm does not require significant costs for the organization of work, so it was more efficient than sequential partitioning over the entire range of changes in *n*. We also note that with the stabilization of the value of *k* starting from $n = 90$, the algorithm on all graphs gives areas of linear growth corresponding to a constant *k*. By decreasing the value of *k*, there is the formation of "steps" in the decrease of execution time when increasing n and the transition to the next section, which is parallel to the previous one (fig. 13,*b*). This confirms the assumption that the execution time grows linearly as *n* increases. It can also be noted that the fragments of linear time growth for the sequential and parallel versions have a different inclination angle to the horizon. At the same time, since this angle is larger for the sequential version, the difference in execution time between the versions increases as *n* increases. Therefore, it can be stated that when the *n* increases, the benefit of parallel versions increases, too, striving to reach the theoretical value – by a factor of two.

Discussion. Let's compare the results obtained with those of the closest studies. In [24], an algorithm for equal chord partition into segments called the Iso-Level Algorithm (ILA) was presented based on the grid approximation of the distance function between two curve`s points. The computational complexity of this algorithm is $O(n \cdot m^3)$ (where m is the number of discretization grid lines for the function approximation). According to [24], $m > n$ and, therefore, $O(n \cdot m^3) > O(n^4)$, which is a very slow. In support of this, the experiments on breaking the curves presented in this paper were carried out for small values of *n* - up to 12 segments. Unlike the algorithm [24], the presented algorithm works well for larger values of *n* when its computational complexity becomes proportional to *n*. The proposed algorithm depends on the curve's shape because the lower limit of *n* values depends on it. Starting from that limit, the intersection with the curve will be unique for all circle positions. The shape of the curve also affects the number of iterations when choosing the option of the circle`s move – direct, reverse, or two-way.

The ECLD algorithm, presented in [26], solves a slightly different task – finding the points on a curve for a predefined chord length. However, the idea based on it can be applied to modify the algorithm presented. This idea is interesting because it does not require finding intersection points and, therefore, does not require solving a nonlinear equation. It can also be applied in cases where the intersection between a circle and a curve is multivalued. Thus, research on such an algorithm modification can be considered promising.

Conclusions. The article examines the problem of a flat curve partition into segments equal in chord length. A new approach to solving this problem in the "classical" formulation is proposed based on the intersection of a curve with a circle of constant radius. Three versions for partitioning the curve by moving the circle are considered. The components of the algorithm are presented – procedures for initiating the circle's radius, partitioning the curve by a circle, and the complete procedure for finding an equal-chord partition. The experimental part consisted of a study of the dependence of the iterations required to ensure the tolerance of the equal-chord partition on the number of segments for different algorithm options, the effect of increasing the tolerance on the increase in iterations, and the influence of the numerical solve method`s tolerance. Experiments were also conducted to measure the executing time for sequential and parallel versions in various segmentation degrees.

As a result of the research, it was found that the proposed algorithm is quite suitable for the equal chord segmentation of flat parametric curves in a wide range of segment values. It was established that with an increase in the degree of segmentation, the iterations to achieve the necessary tolerance decreases, reaching stabilization values significantly less than the number of segments. This led to the fact that starting from a particular value of the partition degree, its execution time increased linearly. The two-way version of the algorithm was the most promising for real applications, as it is more stable and flexible. This version is

suitable for parallel execution by two processes or threads. The two-threaded version showed the best performance of all the algorithm versions. The disadvantages of the presented algorithm should include, first of all, the limitation of the lower limit of segment number because with a small number of them, the radius of the partition circle increases, which leads to the presence of several intersection options and the need to analyze them. Another disadvantage is that obtaining the points by way of the intersection requires solving the nonlinear equation, which depends on the representation of the curve and can be pretty tricky, even for numerical methods.

Summarizing the conducted research and developing the proposed approach to the equal chord segmentation of curves, it is possible to highlight the following promising directions for further study:

- study of the conditions for the unambiguity of the intersection between a flat curve and a circle in a given direction;
- developing the algorithm for partitioning in conditions of multivalued solution of the intersection equation;
- an extension of the algorithm for the case of spatial parametric curved lines;
- applying approaches to finding partition points that do not require solving the intersection equation or simplifying it;
- applying the algorithm to curves given by a discrete sequence of points.

Data availability statement. The data and codes that support this study are available in [28].

References

1. Shpitalni M., Koren Y., Lo C.C. Realtime curve interpolators. *Computer-Aided Design*. (1995). 26(11). P. 832–838. https://doi.org/10.1016/0010-4485(94)90097-3
2. Panagiotakis C., Tziritas G. Snake terrestrial locomotion synthesis in 3D virtual environments. *The Visual Computer*. 2006. 22. P. 562–576. https://doi.org/10.1007/s00371-006-0035-1
3. Ramer U. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*. (1972). 1(3). P. 244–256. https://doi.org/10.1016/S0146-664X(72)80017-0
4. Douglas D., Peucker Th. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*. 1973. 10(2). P. 112–122. https://doi.org/10.3138/FM57-6770-U75U-7727
5. Phisannupawong T., Damanik J.J., Choi H.L. Aircraft Trajectory Segmentation-based Contrastive Coding: A Framework for Self-supervised Trajectory Representation. *arXiv preprint*. 2024. https://doi.org/10.48550/arXiv.2407.20028
6. Rutkowski W.S. Shape segmentation using arc/chord properties. Comput. *Graphics Image Processing*. 1981. 17(2). P. 114–129. https://doi.org/10.1016/0146-664X(81)90020-4
7. Phillips T.-Y., Rosenfeld A. A method of curve partitioning using arc-chord distance. *Pattern Recognition Letters*. 1987. 5(4). P. 285–288. https://doi.org/10.1016/0167-8655(87)90059-6
8. Marji M., Klette R., Siy P. Corner Detection and Curve Partitioning Using Arc-Chord Distance. *IWCIA 2004. Lecture Notes in Computer Science*. 2004. 3322. P. 512–521. https://doi.org/10.1007/978-3-540-30503-3_37
9. Kolesnikov A. ISE-bounded polygonal approximation of digital curves. *Pattern Recognit. Lett*. 2012. 33(10). P. 1329–1337. https://doi.org/10.1016/j.patrec.2012.02.015
10. Williams C. M. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing*. 1978. 8(2). P. 286–293. https://doi.org/10.1016/0146-664X(78)90055-2
11. Dunham J. G. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1986. 8(1). P. 67–75. https://doi.org/10.1109/TPAMI.1986.4767753
12. Horst J., Beichl I. Efficient piecewise linear approximation of space curves using chord and arc length. *Proceedings of the SME Applied Machine Vision '96 Conference*. 1996. https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=820563 (accessed 13.08.2024)
13. Kalaivani S., Ray B.K. A heuristic method for initial dominant point detection for polygonal approximations. *Soft Computing*. 2019. 23. P. 8435–8452. https://doi.org/10.1007/s00500-019-03936-1
14. Hu R., Watt S. Optimization of point selection on digital ink curves. *Proc. 13th International Conference on Frontiers in Handwriting Recognition, Bari, Italy, Sept. 18-20*. 2012. P. 525–530. https://doi.org/10.1109/ICFHR.2012.252
15. de Figueiredo L. H. Adaptive sampling of parametric curves. *Graphics Gems V*. 1995. P. 173–178. https://doi.org/10.1016/B978-0-12-543457-7.50032-2

16. Zhong W., Luo X., Chang W., etc. A real-time interpolator for parametric curves. *International Journal of Machine Tools and Manufacture*. 2018. 125. P. 133–145. https://doi.org/10.1016/j.ijmachtools.2017.11.010
17. Wei J., Sun C., Zhang X. J., etc. An efficient and accurate interpolation method for parametric curve machining. *Scientific Reports*. 2022. 12. 16000. https://doi.org/10.1038/s41598-022-20018-9
18. Han X.T., Zhu K.F., Wang X.B. A hash approach to refine CNC computation of arc length and parameter of NURBS with high efficiency and precision. International Journal of Precision Engineering and Manufacturing. 2024. 25. P. 1243–1256. https://doi.org/10.1007/s12541-024-00976-y
19. Chalkis, A., Katsamaki, Ch., Tonelli-Cueto, J. On the error of random sampling uniformly distributed random points on parametric curves. *Proceedings of the 2022 International Symposium on Symbolic and Algebraic Computation (ISSAC 22)*. 2022. P. 273–282. https://doi.org/10.1145/3476446.3536190
20. Pagani L., Scott P.J. Curvature based sampling of curves and surfaces. *Computer Aided Geometric Design*. 2018. 59. P. 32–48. https://doi.org/10.1016/j.cagd.2017.11.004
21. Hernández-Mederos V., Estrada-Sarlabous J. Sampling points on regular parametric curves with control of their distribution. *Computer Aided Geometric Design.* 2003. 20 (6). P. 363–382. https://doi.org/10.1016/S0167-8396(03)00079-7
22. Frolov O.V., Losev M.U. Modeling of asymptotically optimal piecewise linear interpolation of plane parametric curves. *Radio Electronics, Computer Science, Control*. 2021. 3. P. 57–68. https://doi.org/10.15588/1607-3274-2021-3-6
23. Lopez M.A., Reisner S. A note on curves equipartition. *arXiv preprint*. 2008. https://doi.org/10.48550/arXiv.0707.4298
24. Panagiotakis C., Athanassopoulos K., Tziritas G. The equipartition of curves. *Computational Geometry*. 2009. 42 (6–7). P. 677–689. https://doi.org/10.1016/j.comgeo.2009.01.003
25. Panagiotakis C., Tziritas G. Any dimension polygonal approximation based on equal errors principle. *Pattern Recognit. Lett*. 2007. 28. P. 582–591. https://doi.org/10.1016/j.patrec.2006.10.005
26. Wang X.-R., Wang Z.-Q., He P., etc. The high-energy micro-arc spark—computer numerical control deposition of planar NURBS curve coatings. *Int J Adv Manuf Tech*. 2016. 87. P. 3325–3335. https://doi.org/10.1007/s00170-016-8698-x
27. Divide. https://docs.mcneel.com/rhino/8/help/en-us/commands/divide.htm (accessed 20.10.2024)
28. Frolov O. Equipartition algorithm for a flat parametric curve based on the intersection between it and a moving circle, Oct 2024. https://github.com/froleg/equipartition (accessed: 11.11.2024)

Received 11.11.2024

Oleg Frolov,
PhD, Associate Professor, Department of Information Systems,
Simon Kuznets Kharkiv National University of Economics, Ukraine.
https://orcid.org/0000-0002-2250-5857
frolgx@gmail.com

MSC 68U05

Oleg Frolov

Equipartition Algorithm for a Flat Parametric Curve Based on the Intersection Between it and a Moving Circle

Simon Kuznets Kharkiv National University of Economics, Ukraine
Correspondence: frolgx@gmail.com

Introduction. The problem of discretization of continuous geometric objects is one of the most common problems of computational geometry. Many applications in all different fields, such as computer vision, robotics, signal processing, curve simplification in computer graphics applications, geographic information systems, and digital manufacturing applications, are based on the discretization and segmentation of plane curves, which are basic geometric objects. These methods mainly aim to solve the problem of dividing the curve into segments with the same characteristics or to minimize a predetermined error. The condition of partitioning the curve into points when the lengths of the chords connecting the segments are equal is an additional factor interesting from the point of view of practical applications. It allows, for example, to simplify the reproduction of a curve on CNC machines thanks to the constancy of the tool feed speed [1] or the reproduction of the movement of an object based on a video recording [2].

The purpose of the paper is to develop new algorithms for partitioning flat parametric curves under the condition of equality of chords (chord length connecting segments of the partition) given the two outside points included in the first and last segment and given a number of segments.

Results. The problem of partitioning a curve in a parametric vector form on the Euclidean plane into segments equal in chord length having the formulation of [23, 24] was considered. A method of partitioning a flat parametric curve into equal-chord segments by crossing a circle of constant radius with the subsequent movement of the circle's center to the intersection point is proposed. The problem of the multivalued solution of the intersection equation was considered, which complicates the application of this method. This circumstance limits the use of circular partitioning by the lower limit of the values of the number of segments. The proposed algorithm was presented in pseudocode and described. It consists of the following procedures: the procedure for the initial initialization of the radius of a circle based on a partition with a uniform distribution by a parameter, procedures for partitioning the curve by a circle for different directions of the circle`s move (direct, reverse, two-way); the procedure for obtaining an equal-chord partition with a specified tolerance of determining the chord length. For the real curve`s example, experiments were conducted on its equipartition by this algorithm, implemented in the Julia programming language. It was established that with an increase in the degree of discretization of the value of the curve, the number of iterations required to achieve the specified accuracy stabilizes. This leads to a linear dependence of the partition execution time with an increase in the number of segments. It was found that when the accuracy of the partition is increased, the number of iterations increases slightly compared to the increase in accuracy.

Conclusions. As a result of the research, it was found that the proposed algorithm is quite suitable for the equal chord segmentation of flat parametric curves in a wide range of segment values. The two-way version of the algorithm was the most promising for real applications, as it is more stable and flexible. This version is suitable for parallel execution by two processes or threads. The two-threaded version showed the best performance of all the algorithm versions. The disadvantages of the presented algorithm should include, first of all, the limitation of the lower limit of number of segments because with a small number of them, the radius of the partition circle increases, which leads to the presence of several intersection options and the need to analyze these options. Another disadvantage is that obtaining the points by way of the intersection requires solving the nonlinear equation, which depends on the representation of the curve and can be pretty tricky, even for numerical methods.

Keywords: pseudocode, iteration, computational complexity, segmentation, chord, intersection equation.

УДК 519.85, 004.92

О.В. Фролов

Алгоритм рівноланкового розбиття плоскої параметричної кривої на основі її перетину з рухомим колом

Харківський національний економічний університет імені Семена Кузнеця, Харків
Листування: frolgx@gmail.com

Вступ. Проблема дискретизації неперервних геометричних об’єктів – це одна з найпоширеніших проблем обчислювальної геометрії. Ця проблема має велику кількість застосувань в усіх різних сферах, таких як комп’ютерний зір, робототехніка, обробка сигналів, спрощення кривих у застосунках комп’ютерної графіки, геоінформаційних системах та застосунках для цифрового виробництва, засновані на дискретизації та сегментації плоских кривих, які є базовими геометричними об’єктами. Основна мета методів дискретизації полягає у тому, щоб вирішити задачу розбиття кривої на сегменти з однаковими характеристиками або мінімізувати заздалегідь визначену помилку. Умова розбиття кривої точками при рівності довжин хорд стягуючих сегменти є додатковим фактором цікавим з точки зору практичних застосувань. Вона дозволяє, наприклад, спростити відтворення кривої на станках із ЧПУ завдяки сталості швидкості подачі інструмента [1], або відтворення руху об’єкта за відеозаписом [2].

Мета роботи. Розроблення нових алгоритмів розбиття плоских параметричних кривих за умови рівності ланок (довжин хорд стягуючих сегменти розбиття) при поданих двох крайніх точках, що входять до першого та останнього сегменту, та заданої кількості сегментів.

Результати. Розглядалась задача розбиття кривої, заданої у параметричній векторній формі на евклідовій площині, на рівні за довжиною хорди сегменти у «класичному» формулюванні [23, 24]. Запропоновано метод розбиття плоскої параметричної кривої на рівні за хордою сегменти перетином кола сталого радіусу з наступним переміщенням центру кола у точку перетину. Було розглянуто проблему багатозначності розв'язку рівняння перетину, що ускладнює застосування цього методу. Ця обставина обмежує застосування розбиття колом нижньою границею значень кількості сегментів. Було представлено у псевдокоді і описано запропонований алгоритм, а саме: процедуру первісної ініціалізації радіусу кола на основі розбиття з рівномірним розподілом за параметром; процедури розбиття кривої колом для різних напрямів переміщення кола (прямий, зворотний, двосторонній); процедуру отримання рівноланкового розбиття із заданою точністю визначення довжини хорди. На прикладі реальної кривої лінії проведено експерименти із її розбиття алгоритмом, реалізованим на мові програмування Julia. Встановлено, що із збільшенням ступеня дискретизації кривої значення кількості ітерацій, необхідних для досягнення заданої точності, стабілізується. Це призводить до лінійної залежності часу виконання розбиття зі збільшенням кількості сегментів. Виявлено, що при підвищення точності розбиття відбувається незначне у порівняння із зростанням точності збільшення показника кількості ітерацій.

Висновки. У результаті проведених досліджень встановлено, що запропонований алгоритм цілком придатний для сегментації плоских параметричних кривих за умови рівності довжин хорд у широкому діапазоні значень кількості сегментів. Для реальних застосувань найбільш перспективною виявився двосторонній варіант алгоритму, як більш стабільний та гнучкий. Цей варіантf алгоритму дозволяє виконувати розбиття паралельно двома процесами або потоками. Версія з двома потоками продемонструвала найкращі показники часу виконання серед всіх версій алгоритму. До недоліків представленого алгоритму слід віднести, насамперед, обмеження нижньої границі значень кількості сегментів, адже при їх невеликої кількості збільшується радіус кола розбиття, що призводить до наявності декількох варіантів перетину та необхідності аналізу цих варіантів. Недоліком також є те, що для отримання точок розбиття на основі перетину з колом необхідно розв'язувати нелінійне рівняння, яке залежить від представлення кривої, і може бути досить складним, навіть, для чисельних методів.

Ключові слова: псевдокод, ітерація, обчислювальна складність, сегментація, хорда, рівняння перетину.