Interval Methods for Kinetic Simulations

Leonidas J. Guibas* Menelaos I. Karavelas*

Graphics Lab., Computer Science Dept., Stanford University
Stanford, CA 94305, U.S.A.
{guibas, karavelas}@cs.stanford.edu

Abstract

We propose a speed-up method for discrete-event simulations, including sweep-line or -plane techniques, requiring the repeated calculation of the times at which certain discrete events occur. Instead of calculating these event times precisely, we use interval methods to obtain less expensive approximations that may still be adequate for the simulation. This can happen because some events get descheduled before they actually happen, or because event time comparisons can be resolved using only information about their bounding intervals. The computed intervals are refined as necessary, when greater accuracy is needed. For geometric objects described by polynomials, including moving objects on polynomial trajectories, the proposed method is shown to give a speed-up roughly proportional to the degree of the polynomials.

1 Introduction

Discrete-event simulation is commonly used by geometric algorithms to solve a variety of problems, in both the static and kinetic settings. Many classic geometric algorithms are of the sweep-line type (or sweep-plane, etc, in higher dimensions), in which a sweep is used to reduce a static problem in a given dimension to a dynamic problem in one less dimension. A sweep algorithm typically maintains an event queue, where the event times are the moments when the sweep line needs to stop and perform updates to the data structures it maintains, including the event queue itself. A famous classic example of such an algorithm is the Bentley-Ottmann line-sweep algorithm [3] for detecting all intersections among a set of line segments in the plane. All kinetic data structures (KDSs) [2, 4], which are data structures aimed at dealing with geometric objects in motion, are also based on an event queue, where the event times are certificate failures associated with the proof of correctness of a computation of the KDS attribute of interest. When a certificate fails, the proof, and with it the event queue, needs to be updated. In both cases a priority queue on event times is maintained and the algorithm repeatedly advances the clock to the next event and updates the queue. We will refer to both of these scenarios as *kinetic simulations*, because they involve the continuous evolution of a system punctuated by discrete events.

The calculation of an event time is frequently a non-trivial computational task. For example, it may involve computing the intersection of three-surfaces in the sweep-plane case, or the time when moving points become coplanar or co-spherical, in the kinetic case. The cost of such computations cannot always be justified in terms of the final result that needs to be computed. For example, almost all kinetic simulations involve the *de-scheduling of events* — these are events that will not happen because the associated certificates were removed from the proof, and the computational resources that went into their event-time calculation will be wasted. In general, of course, it is hard to know, at the time an event is scheduled, that it will be de-scheduled at some future time in the kinetic simulation. Moreover, in many sweep and kinetic problems the exact time when events happen may not be needed, as long as we can guarantee that the correct sequence of events will be generated. This is exactly the case in the classic Bentley-Ottmann sweep, where the output is a purely combinatorial list of intersecting pairs of segments.

The goal of this paper is to present an approach to reducing the cost of event-time calculations in kinetic simulations, through the use of interval methods, akin to interval arithmetic [1]. Instead of exact event times, we will focus on time intervals guaranteed to contain one or more events. The key intuition is that we do not need to know very precisely events scheduled to happen far into the future. We want to devote computational resources to refining the intervals associated with these events only as they get closer to the present time in the simulation. By using intervals for all event-queue operations as well, we are often able to resolve comparisons between event times without further refinement of the associated intervals. The result is that we are able to generate the correct sequence of events for the kinetic simulation, but at a substantial savings in the cost of the event-time calculations.

To realize and evaluate experimentally this idea, we concentrate in this paper on events whose event times can be calculated by solving polynomial equations. Since almost without exception kinetic certificates are low degree polynomial functions of attributes (e.g., positions) of a small number of the moving bodies (e.g. the `CCW` or `InCircle` tests), this restriction covers the case where the motions themselves

*Work by the authors is supported by National Science Foundation grant CCR-9623851 and by U.S. Army Research Office MURI grant DAAH04-96-1-0007, and by a grant from the U.S.-Israeli Binational Science Foundation.

are polynomial (in the sweep case an equivalent condition is that the curves or surfaces involved are polynomial functions). Polynomials are an attractive class of functions to consider, because efficient root isolation methods for them have been well studied. We specifically make use of the technique of *standard sequences* to determine intervals containing the roots (event times) of interest. Furthermore, more general functions can often be well approximated by polynomials in certain ranges (e.g., Taylor expansions). Our techniques can be extended to this case by including additional certificates whose failure indicates that a particular approximation is invalid and a new polynomial approximation needs to be generated.

The remaining sections of the paper are as follows. In Section 2 the overall framework of a kinetic simulation is described in more detail. In Section 3 we present the algebraic and analytic tools used to develop our algorithm, i.e., we present the notion of standard sequences, as well as Sturm's and Bolzano's theorems. Then in Section 4 we present the details of the interval-based kinetic scheduler, including the refinement and update policies for intervals isolating polynomial roots, and the priority queue maintenance using intervals as opposed to exact event times. In Section 5 we provide an informal theoretical justification of the advantages of our approach. In Section 6 we provide a framework for comparing the interval and ordinary schedulers and present empirical data on the superiority of the interval approach. In Section 7 we discuss a trade-off between degree and number of pieces for splined motion trajectories. Finally we conclude in Section 8 with some additional remarks and plans for further work.

2 Kinetic Simulations

The inner loop of a kinetic simulation is the maintenance of the associated event queue. The entries of the event queue are the future failure times of the certificates currently in the kinetic proof — we will call these the *active* certificates. At each step of the kinetic simulation the next certificate to fail is obtained from the priority queue and the kinetic proof is updated to accommodate the altered state of the world. As a result, typically a number of active certificates leave the proof (and event queue) and a number of other new certificates enter the proof and become active. In a well-designed KDS the number of certificate deletions and insertions per certificate failure is small (this is the concept of a *responsive* KDS [4]).

Each certificate is typically a simple algebraic inequality on the positions/poses of a small number of features of the moving objects. In fact, in most kinetic simulations only a small number of different types of certificates are ever used (for example, a kinetic Voronoi/Delaunay simulation for point sites can be done using only the `InCircle` test).

The above considerations motivate the following formulation of the problem: let $\mathcal{S}$ be a set of polynomials $\{f_1(t), f_2(t), \ldots, f_k(t)\}$ (corresponding to the certificates in the KDS), the real roots of which represent possible events in our simulation. There is a notion of a current time t_0 and we are interested in quickly finding the time t_1, which is the smallest root of any of the f_i greater that t_0. Then we perform some changes in the set $\mathcal{S}$ and advance in time by setting $t_0 \leftarrow t_1$.

The naive solution to this problem is the following: for each polynomial f_i compute all its roots to the required precision, discard those that are complex and insert its smallest real root greater than the current time into the event queue (we can think of the event queue as a priority queue implemented using a heap). Some methods for computing the roots of a polynomial are the Jenkins/Traub method [6], the eigenvalue method [13, 11] in which we construct the companion matrix of the polynomial and compute its eigenvalues, Muller's and Laguerre's methods [11] and a more recent method by Lang and Frenzel [9]. Among these methods the last one, although very accurate for high degree polynomials, is rather expensive. Muller's, Laguerre's and the Jenkins/Traub methods, however, are not stable enough for polynomials of degree greater than 60 or so. Thus we decided to adopt the eigenvalue method for both our theoretical analysis as well as the implementation of the KDS. As already mentioned, our goal is to avoid spending resources in computing real roots that correspond to events that may never happen, or complex roots that are of no interest for our simulation. In addition, we want to compute the real roots of the polynomials only to such accuracy as required to resolve root comparisons and determine which polynomial among the two being compared has the earliest failure time.

The approach that we employ in this work is to use the *standard sequence* [5] of a polynomial f in the queue to maintain an ordered list of intervals that contain and isolate its real roots. The leftmost among these intervals is the one that represents the certificate for f in the priority queue. The comparison of two polynomials in the queue is done by comparing the intervals and splitting them as necessary. This process of resolving comparisons, as well as the forward stepping in time related to the update of the current time t_0, cause us to refine these interval lists and obtain tighter bounds on the roots of the polynomials.

The main advantages of this approach are as follows. First of all, operations on the interval list are only performed when needed, that is when the information obtained so far about the roots is not sufficient to resolve root comparisons, and thus to determine the relative priority of the two polynomials in the queue. These operations are focussed on the smallest root of each polynomial, rather than all the roots at the same time, thus avoiding spending computation time on possible events that may eventually not happen. Moreover, if we were to use a symbolic algebra system for performing computations, then our algorithm could be implemented with exact operations — unlike the naive method which must always resort to numerical techniques.

3 Mathematical Preliminaries

Let $y = \{y_1, y_2, \ldots, y_m\}$ be a finite sequence of non-zero numbers. We define the *number of variations in sign* of y to be the number of indices i, $1 \leq i \leq m-1$, such that $y_i y_{i+1} < 0$. If $y = \{y_1, y_2, \ldots, y_m\}$ is an arbitrary sequence of numbers, then we define the number of variations in sign of y to be that of the subsequence y' obtained by dropping the zeros in y. For example the number of sign variations of $\{4.5, 0, 0, 0.5, -1.3, 0, 10^{-30}, 4, -200\}$ is 3.

Let now $f(x)$ be a polynomial of positive degree with real coefficients. Then the sequence of polynomials $\{f_0(x),$

$f_1(x)$, ..., $f_s(x)$} defined by repeated division as:

$$\begin{aligned} f_0(x) &= f(x) && (1)\\ f_1(x) &= f'(x) && (2)\\ f_0(x) &= q_1(x)f_1(x) - f_2(x) && (3)\\ &\vdots\\ f_{i-1}(x) &= q_i(x)f_i(x) - f_{i+1}(x) && (4)\\ &\vdots\\ f_{s-1}(x) &= q_s(x)f_s(x) \quad (\text{i.e., } f_{s+1} = 0), && (5) \end{aligned}$$

where the degrees of the f_i monotonically decrease, is called the *standard sequence* for $f(x)$ [5]. As it can easily be verified, the $f_i(x)$, $i \geq 2$ are obtained by modifying Euclid's algorithm for finding the g.c.d. of $f(x)$ and $f'(x)$ in such a way that the last polynomial obtained at each stage is the negative of the remainder[1] in the division process:

$$f_{i+1}(x) = -f_{i-1}(x) \bmod f_i(x), \quad i = 1, \ldots, s-1. \qquad (6)$$

In view of the above, $f_s(x)$ is the g.c.d. of $f(x)$ and $f'(x)$. In particular, if $f_s(x)$ is a constant polynomial then $f(x)$ has no multiple roots. Moreover, if $f_s(x)$ is not a constant polynomial, then if $g_i(x) = f_i(x)/f_s(x)$, $0 \leq i \leq s$, then $g_0(x)$ has the same roots as $f(x)$, but now all the roots of $g_0(x)$ are simple. Moreover the sequence $\{g_i(x)\}_{i=0}^{s}$ is the standard sequence for $g_0(x)$.

Using the notion of standard sequences just described we are ready to state *Sturm's theorem*, which addresses the problem of counting the number of real roots of a polynomial in an interval of the real line:

STURM'S THEOREM[2]. *Let $f(x)$ be a polynomial of positive degree with real coefficients and let $\{f_0(x) = f(x), f_1(x) = f'(x), \ldots, f_s(x)\}$ be the standard sequence for $f(x)$. Assume $[a, b]$ is an interval of the real line such that $f(a) \neq 0$, $f(b) \neq 0$. Then the number of distinct real roots of $f(x)$ in (a, b) is $V_a - V_b$ where V_c denotes the number of variations in sign of $\{f_0(c), f_1(c), \ldots, f_s(c)\}$.*

It is shown in [14] that the roots of $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$ lie in $[-\alpha, \alpha]$, where

$$\alpha = 1 + \frac{\max\{|a_{n-1}|, \ldots, |a_0|\}}{|a_n|}. \qquad (7)$$

Hence, if $\mu = \alpha + \varepsilon$ (for some $\varepsilon > 0$) and $\{p_0(x) = p(x), p_1(x) = p'(x), \ldots, p_s(x)\}$ is the standard sequence of $p(x)$, then the total number of distinct real roots of $p(x)$ is $V_{-\mu} - V_{\mu}$, where, as before, V_c is the number of variations in sign of $\{p_0(c), p_1(c), \ldots, p_s(c)\}$.

Another very well-known theorem that will be of use is *Bolzano's theorem*:

BOLZANO'S THEOREM. *Let $f(x)$ be a continuous real valued function in the interval $[a, b]$, and assume that $f(a)$ and $f(b)$ have opposite signs, i.e., $f(a)f(b) < 0$. Then there is at least one c in the open interval (a, b) such that $f(c) = 0$.*

[1] An algorithm for finding such division remainders for two polynomials with real coefficients can be found in [8].

[2] Sturm's theorem holds true for polynomials with coefficients in any real closed field R. The statement of the theorem that we present deals only with the case of real coefficients (see [5] for the generic statement).

Based on the above two theorems we can define the two primitives we will use, $T_f(a, b)$ and $S_{f,N}(a, b)$. The primitive $T_f(a, b)$ counts the number of distinct real roots of the polynomial $f(x)$ in (a, b), provided that $f(a)f(b) \neq 0$. The primitive $S_{f,N}(a, b)$ gives the number of (distinct) real roots of $f(x)$ in (a, b) provided that $f(a)f(b) \neq 0$ and that the number of real roots of $f(x)$ in (a, b) does not exceed N.

The purpose of introducing the second primitive is that if we know that $N = 1$ and that $f(x)$ has only simple roots, if any, in (a, b), then we can determine the number of real roots of $f(x)$ in (a, b) by simply checking whether $f(a)f(b) < 0$ or not. This test is much cheaper, especially for high degree polynomials, than the one suggested by *Sturm's Theorem*. Note that if $f(x)$ has only simple roots in (a, b) and $\deg f(x) \geq 2$, then $T_f(a, b)$ is equivalent to $S_{f,N}(a, b)$ for $N = \deg f(x)$.

Finally, given two polynomials $f(x)$ and $g(x)$ and an interval $[a, b]$ we can determine, using the above machinery, whether they have a common real root in $[a, b]$. This can be done by computing the g.c.d. h of f and g, and then using the above mentioned predicates to determine if h has a real root in $[a, b]$.

4 The Interval-based Kinetic Scheduler

In order to reduce the cost of event time calculations in the event queue, we keep intervals that contain one or several of those event times. This choice enables us to avoid wasting computing resources when the comparison between two event times can be resolved by considering their corresponding intervals. Moreover, by our approach, we avoid spending computing time on events that are scheduled to happen in the future, and which may be de-scheduled before their time of occurrence (e.g., because the "flight" plans of the objects in the kinetic simulation have changed in the meantime, or because of other changes in the kinetic proof). By using intervals we are able to focus only on the first event time associated with a certain certificate that is greater than the current time; computations that have to do the remaining event times related to the certificate in question are postponed until later, when they are actually needed. Clearly, the better estimates we have for our event times, the easier the comparisons are. For this purpose, if during the process of comparing event times based on their interval representation we get finer bounds on these times we store and use these to facilitate other comparisons that are performed during the process of updating the event queue.

Let $f(x)$ be a polynomial that represents one of the certificates in our kinetic simulation. With each such polynomial we associate an ordered interval list (a_1, b_1), ..., (a_m, b_m) such that $b_i \leq a_{i+1}$, $i = 1, \ldots, m-1$, $f(a_i)f(b_i) \neq 0$ and $T_f(a_i, b_i) > 0$, $i = 1, \ldots, m$. All real roots of $f(x)$ greater than the current time t_c are contained in one of these intervals (initially the list consists of a single interval containing all the real roots of $f(x)$). Suppose now that we want to determine, among two polynomials $p(x)$ and $q(x)$, which is the one that corresponds to the earliest event time, i.e., which is the one that has the smallest real root (greater than t_c). We can also think of these event times as the *priorities* of $p(x)$ and $q(x)$ in the event queue; the question, therefore, is which, among $p(x)$ and $q(x)$, has the greater priority. Let (a, b) and (c, d) be the leftmost intervals in the lists of $p(x)$ and $q(x)$, respectively. We can assume that the roots of these polynomials are simple, since otherwise we can replace the polynomials, as described in the previous section, with others that have only simple roots. Without loss of generality

we can also assume that $a \leq c$ (otherwise we can interchange the roles of $p(x)$ and $q(x)$). The procedure that we follow to determine the relative priority of $p(x)$ and $q(x)$ is the following (note that we test a condition only if all the previous ones have failed) :

1. if $b \leq c$ then the smallest root of $p(x)$ will be smaller than the smallest root of $q(x)$, and thus $p(x)$'s priority will be greater than that of $q(x)$.
2. if $p(x)$ has any roots in $(a, c]$ then $\mathsf{priority}(p(x)) > \mathsf{priority}(q(x))$.
3. if $b \leq d$ and $q(x)$ has all its roots in $[b, d)$, then the priority of $p(x)$ is greater than that of $q(x)$.
4. if $b > d$ and all the roots of $p(x)$ are in $[d, b)$, then $\mathsf{priority}(p(x)) < \mathsf{priority}(q(x))$.
5. if both polynomials have some of their roots in the interval (c, k), where $k = \min\{b, d\}$, then we need to employ a subdivision-like approach: we split (c, k) in the middle and check if the real roots of $p(x)$ and $q(x)$ are distributed in such a way in the two resulting intervals that we can directly determine their relative priority (e.g., if $p(x)$ has roots in $(c, \frac{c+k}{2})$ and $q(x)$ does not then $\mathsf{priority}(p(x)) > \mathsf{priority}(q(x))$). We recursively continue this subdivision process until we can determine which of the two polynomials has higher priority.

If the smallest roots of the two polynomials are not the same then the subdivision procedure terminates; on the other hand if they share that root then something else has to be done. What we do is the following test: if the two polynomials have only one root in the interval of interest we check to see if that root is a common one. To do so we compute their g.c.d. and check if it has a root in the interval of interest. The natural question that arises is how we can be sure that the polynomials will have only one root in that interval.

At various points during the algorithm an interval (α, γ), associated with a polynomial $f(x)$, needs to be split in two parts (α, β) and (β, γ). If $f(\beta) \neq 0$, and $f(x)$ has roots in both intervals then we replace (α, γ) with (α, β) and (β, γ). If only one of the two intervals contains roots of $f(x)$ then we simply update the corresponding endpoint. If $f(\beta) = 0$, then find a point β' to the left or to the right of β, such that $f(\beta') \neq 0$ and do the splitting using that point. Since we have to do these interval splits anyway in order to determine the relative priority of the two polynomials we basically get for free better bounds on the roots of both polynomials. However, these better approximations of the roots are only computed when the information obtained so far is not sufficient to determine which polynomial is of higher priority. Moreover, splitting the intervals results in interval lists that will eventually contain only a single real root of the polynomial in question, which is important for determining if two polynomials have a common real root.

So far we did not properly take into account that we have a current time t_c and that we are interested only in real roots larger than that time. In addition, this current time is represented as the root of a polynomial $c(x)$. Therefore we do not know it exactly, but rather we have an interval (α_c, β_c) in which it lies. This interval is the first interval in the interval list associated with $c(x)$. We can actually assume that the current time is the only root of the associated polynomial in that interval, since otherwise we can use the midpoint of the interval to split it; we can continue this recursively until we get a list in which the first interval contains only one root, which is going to be t_c. To take account of this fact, the interval list of a polynomial $p(x)$ needs some preprocessing which has to do with discarding the roots that are smaller than t_c. This procedure is as follows:

1. discard all of the intervals (a_i, b_i) such that $b_i \leq \alpha_c$ (if any) and then renumber the remaining ones.
2. if $\beta_c \leq a_1$ there is nothing more to be done: we have already kept all those roots that are greater than t_c.
3. if $\beta_c > a_1$, we check if t_c is in $(\alpha_c, a_1]$; in that case we simply update our bounds for t_c.
4. if $b_1 \leq \beta_c$ then
   (a) if t_c is in $[b_1, \beta_c)$, then delete (a_1, b_1), update α_c, and proceed in the same manner with the new (a_1, b_1).
   (b) if t_c is not in $[b_1, \beta_c)$, then it has to be in (r, b_1), where $r = \max\{a_1, \alpha_c\}$, in which case we update the bounds for t_c and employ the subdivision approach in (r, b_1).
5. if $b_1 > \beta_c$ and $p(t)$ has at least one root in $(r, \beta_c]$, where r is defined as above, then we employ the subdivision approach in $(r, \beta_c]$.

A similar pruning approach can be applied when we want to run the simulation up to a time T_{max}, where T_{max} is assumed not to be an event time. In that case we discard of all the intervals (a_k, b_k) such that $T_{max} \leq a_k$. Let (a_ℓ, b_ℓ) be the last interval in the list; clearly, $a_\ell < T_{max}$. If $b_\ell \leq T_{max}$ then we do nothing; otherwise, we just replace (a_ℓ, b_ℓ) with (a_ℓ, T_{max}), if (a_ℓ, T_{max}) contains any roots of the associated polynomial, or discard it altogether.

On several occasions we have talked about determining whether a polynomial $f(x)$ has real roots in a certain interval (a_i, b_i) or about how many roots there are. The primitive that we can use in these cases is $T_f(a_i, b_i)$, the most generic one among the two we introduced in the previous section. There are instances, however, where we can do better. If we store the number n_i of real roots of the interval (a_i, b_i), then we can use the primitive $S_{f,n_i}(c, d)$, whenever $a_i \leq c \leq d \leq b_i$, which is always the case when we split intervals. This way we can take advantage of the very simple test that the primitive S incorporates if $n_i = 1$ and $f(x)$ has simple roots in (a_i, b_i).

5 A Theoretical Justification

In this section we present a very simple model for the distribution of the event times and perform a worst-case analysis for two methods: our interval-based approach and a method which computes all roots of the certificate polynomials by computing the eigenvalues of the corresponding companion matrix.

In particular, let s be the number of active polynomials and let us assume that each polynomial has d real roots which are random i.i.d. variables in $[0, T_{max}]$, where T_{max} is the time until when we run our kinetic simulation. Let also m be the total number of KDS events occurring during the simulation, and assume that at each event k old certificates (polynomials) leave the event queue and k new certificates enter the queue. Then the expected separation between the event times, i.e., the roots of the polynomials is $\frac{T_{max}}{ds}$, whereas the expected separation between roots of

the same polynomial is $\frac{T_{max}}{d}$. We will also assume that the event queue is implemented using a heap-like structure, so that insertions and deletions in the queue take logarithmic time in the queue size.

The cost of the eigenvalue method is $\mathcal{O}(d^2K)$ where K is the number of iterations performed [13]. In particular, if we want to compute the eigenvalues with accuracy equal to ε, then

$$K = \mathcal{O}(\frac{\log \varepsilon}{\log \max_{1 \le i \le d-1} \frac{|\lambda_{i+1}|}{|\lambda_i|}}) \tag{8}$$

where λ_i are the roots of the polynomial satisfying $|\lambda_{i+1}| \le |\lambda_i|$, $1 \le i \le d-1$. In our case we can assume that $\lambda_i \ge 0$, $\forall i$ (the roots represent time values). In view of our assumption that the roots are evenly distributed and that their distance is $\frac{T_{max}}{d}$ in expectation, we get that

$$\max_{1 \le i \le d-1} \frac{|\lambda_{i+1}|}{|\lambda_i|} \le \frac{d-1}{d} \le \frac{d}{d+1} \tag{9}$$

which implies that

$$K = \mathcal{O}(\frac{\log \varepsilon}{\log \frac{d}{d+1}}) = \mathcal{O}(d \log \frac{1}{\varepsilon}) \tag{10}$$

Since the roots are expected to be $\frac{T_{max}}{ds}$ apart from each other, we only need an accuracy $\varepsilon = \Theta(\frac{T_{max}}{ds})$ which implies that

$$K = \mathcal{O}(d(\log d + \log s)) \tag{11}$$

Hence, the total cost per event using the eigenvalue method is $\mathcal{O}(kd^3(\log d + \log s))$.

Consider now the interval-based method. This method needs $\mathcal{O}(\log ds)$ steps to resolve the comparison between two polynomials (because of the subdivision-like approach that we employ) and at each step the cost is $\mathcal{O}(d^2)$ (this is the cost to compute the predicates $T_f(a,b)$ or $S_{f,N}(a,b)$). Therefore the total cost per event is $\mathcal{O}(kd^2(\log d + \log s))$, which is a factor of d better than that of the eigenvalue method. As we will see in the next section the numerical experiments support the above theoretical calculation.

6 Numerical Experiments

We implemented the algorithm that was described in Section 4 for two KDSs: one for maintaining the Delaunay triangulation (DT) and one maintaining the closest pair (CP) of a set of points moving on the plane. The points are moving on trajectories of the form $(x(t), y(t))$ where both $x(t)$ and $y(t)$ are polynomials of degree d. The coefficients of these polynomials are chosen uniformly from $[-1, 1]$, except their constant term which is chosen uniformly from $[0, 1]$. In the case of the DT the certificates correspond to `InCircle` tests of quadruples of points, hence the degree of the certificates is at most $4d$. In the case of the CP the certificates are polynomials of degree $2d$ or d, that corresponding to comparisons of squared distances for points in the plane or to comparisons of the projections of points along certain (fixed) directions. The details for the certificates for both simulations can be found in [2].

In our examples the number n of moving points is between 10 and 20, whereas the degree d of their motion varies from 2 to 40 in the DT simulation, and from 2 to 80 in the CP simulation. For every pair (n, d) we computed the running times using three different approaches:

(a) the "naive" method, in which we compute all the roots of a polynomial, throw away those that are complex, and use the real ones to resolve the event time comparisons; the roots of a polynomial are computed by constructing the companion matrix and computing its eigenvalues [11, 13],

(b) the interval-based approach that we have already described, and

(c) a hybrid method, in which we isolate the real roots of the polynomial using the predicates $T_f(a,b)$ and $S_{f,N}(a, b)$ and then use a standard root finding technique, like the bisection method [11], to compute the root of the polynomial in each interval.

For each pair (n, d) we started with 10 different initial configurations of points. The experiments were performed on an SGI workstation using an R10000/195 MHz processor.

What we can see from the results, as shown in Figures 1 and 2, is that the eigenvalue method in the DT case, is superior for motion degrees up to 6, whereas for the CP case it is superior for motion degrees up to 13. This should be attributed to the overhead of the interval method due to the evaluation of the standard sequence for each polynomial. However for certificates of higher degree the interval-based method is superior to the eigenvalue method. In fact the data shows that we gain a speed-up factor of order d, where d is the degree of the motion, independently of n. The same can be observed when comparing the eigenvalue and the hybrid methods. The hybrid method, however seems to be a constant factor worse that the interval-based method; this can be attributed to two facts: first of all, the hybrid method computes *all* the real roots of each certificate and not only those that are after the current time; secondly, the roots are computed to greater accuracy than needed in order to resolve the comparisons in the priority queue.

7 Degree vs. events

The cost of a kinetic simulation is an increasing function of the algebraic degree of the motions — more complex motions imply more time-consuming event-time calculations. At the same time, this cost is also an increasing function of the number of events that have to be processed. In this section we consider a trade-off between these two costs. By approximating a high-degree motion by a sequence of lower-degree motions, we can reduce the cost of event-time calculations, while at the same time adding the cost of processing the flight plan updates that must happen at motion segment boundaries. We can actually view this issue backwards as well: if we approximate splined polynomial motions with single polynomials of high degree, then we eliminate events that have to do with flight plan updates, but at the same time we increase the cost of processing the simulation events. Of course kinetic simulations are chaotic systems and there is absolutely no guarantee that the approximated system will have the same sequence of events as the original. Nevertheless, according to our experience, approximations such as the above do preserve the overall character of the simulation as well as various global statistics, and thus are meaningful and useful under certain circumstances.

To examine this trade-off, consider n points moving each along a single parametric polynomial trajectory of degree d_H. We approximate the motion of the points, with motions of lower degree d_L, $d_L < d_H$ in the following manner: we densely sample each higher order trajectory and then

perform a constrained least squares fit to the sampled data. The number of time samples is equal to m and these are uniformly distributed in the time interval of interest; we will discuss the choice of m in the sequel. We impose the constraint that the original and the approximating motions must coincide at the endpoints of each of the time intervals of the approximation (we need to maintain at least C^0 continuity for the splined motion). The interval of approximation is initially the entire time interval for which we run our simulation. We obtain a measure of closeness between the original and approximating trajectories by combining the distances between corresponding points on the two trajectories at each of the sampled times using the L_∞-norm. If this closeness measure fails to be below some prespecified threshold value, then we split the interval of approximation at the sample time of maximum error and recursively repeat the approximation procedure for the two subintervals. The cost of each approximation step is $\Theta(md_L^2)$, dominated by the constrained least squares calculation. The number of times we need to repeat the process will be analyzed below.

In our setting, we want to compute a polynomial segment of degree d_L that approximates a segment of degree d_H to within an error ϵ in the L_∞-norm (or split the interval if that is not possible). We accept a single segment only when we can establish that it meets this criterion. Peetre [10] shows that the error introduced by doing a discrete instead of a uniform polynomial approximation is $\mathcal{O}(1/m^2)$, where m is the number of points used to perform the discrete approximation. Since we calculate the L_∞-norm over a discrete set of m sampled time values, we choose[3] $m = 10/\sqrt{\epsilon}$ so as to guarantee via Peetre a maximum error of $\epsilon/2$ between the discrete and uniform norms. We also make the threshold discussed in the previous paragraph to be $\epsilon/2$. In this way a low degree segment is accepted only when it is known to be within ϵ of the original in the L_∞-norm.

When we approximate a polynomial of degree d_H with one of degree $d_L < d_H$, over the interval $[0, t]$, the error of the approximation is of order $\mathcal{O}(t^r)$, where $r = d_L + 1$. Since we want this error to be at most ϵ, we require that $t = \mathcal{O}(1/\epsilon^{1/r})$. From this estimate for t we see that the number of low degree polynomial pieces required to approximate the original curve in our simulation will roughly be $\mathcal{O}(T_{max}/\epsilon^{1/r})$, where T_{max} is the stop time of the simulation.

The cost of the kinetic simulation, in the model above, is of two types:

1. the cost of resolving comparisons between certificates that have to do with the maintenance of the geometric attribute of interest, and
2. the cost of approximating and updating the trajectories of the moving points.

The first part of the cost is assumed to be equal to the product of the number of events n_{ev} scheduled and descheduled in the priority queue due to the changes in the geometric attribute of interest, times the mean cost of resolving a comparison between event-times. We saw in Section 5 that this cost is $\mathcal{O}(d^2 \log(ds))$, where d is the degree of the certificates and s is the size of the priority queue. Since the degree of the certificates is typically a constant multiple of the degree of the motion, the total cost due to the maintenance of the geometric attribute is $\mathcal{O}(n_{ev} d_L^2 \log(d_L s))$. The second part of the cost has to do with the scheduling and descheduling of events that correspond to changes in the motion, as well as the cost to approximate the original motion with one of lower degree. The cost for these priority queue updates is $\mathcal{O}(\log s)$, yielding a total cost of $\mathcal{O}(n_{mc}[\log s + d_L^2/\sqrt{\epsilon}])$ for this second part, where n_{mc} is the number of events associated with the motion changes. Following the analysis in the previous paragraph, $n_{mc} = \Theta(n\, T_{max}/\epsilon^{1/r})$, where $r = d_L + 1$; thus, the total cost for our simulation is $\mathcal{O}(n_{ev} d_L^2 \log(d_L s) + n\, T_{max}\, [\log s + d_L^2/\sqrt{\epsilon}]/\epsilon^{1/r})$. In many KDSs the size of the proof to be maintained is linear with respect to the number of objects in the simulation, therefore s is taken to be equal to the number of points n. The expression for the total cost of the simulation then becomes $\mathcal{O}(n_{ev} d_L^2 \log(d_L n) + n\, T_{max}\, [\log n + d_L^2/\sqrt{\epsilon}]/\epsilon^{1/r})$.

Assuming that the approximation is accurate enough, the number of events n_{ev} is only a function of d_H. Under this assumption, and for fixed ϵ, it is clear that as d_L decreases, we expect the cost of updating the geometric attribute to decrease and the remaining cost to increase. However, the remaining cost consists of two different parts which behave differently as d_L changes. In particular, for small d_L, since we have a lot of low degree polynomials, the cost of updating the event queue is large; for large d_L, the cost of the approximation dominates. Moreover, we can expect a monotone increase in the cost of the simulation as the error ϵ decreases.

In order to examine the validity of the above analysis we considered $n = 5$ moving points at 10 initial random positions. The geometric attribute that we want to maintain in this experiment is the Delaunay triangulation of the points. The degree of their original trajectory is $d_H = 32$ and the degrees d_L of the approximate trajectories vary from 30 to 2. The stop time for the simulations is $T_{max} = 1$. Our earlier assumptions hold true, namely that the degree of the certificates d is a constant multiple of the degree of the motion d_H or d_L and that the size of the priority queue s is linear in the number of points n. Figure 3 depicts the average running times as a function of the degree of the motion d_L for several errors ϵ. The square corresponds to the simulation where the original trajectory is used. The interval-based approach described in this paper was used to do the simulations.

The main observations are the following :

1. The cost of the simulation increases monotonically as we increase the accuracy. This is in agreement with our model.
2. For fixed accuracy and for decreasing d_L, the cost of the simulation at first decreases, reaches a minimum and then increases. Initially the cost of the simulation is dominated by the cost of computing the approximating motions (due to their large degree); as the degree d_L decreases further, the number of curve pieces needed for the approximation starts to go up and the cost now is dominated by the updates of the motion in the event queue. This, again, is a behavior consistent with the model presented above. The bumps appearing in the graphs should be attributed to the changes in the combinatorial structure of the simulation and the fact that the point where we perform the split in our recursive subdivision of the approximation intervals may not be exactly optimal.
3. The degree of minimal total cost, when we do the approximation, increases as the accuracy increases. Furthermore, for low accuracy, this minimal cost is smaller

[3] The constant 10 here was chosen arbitrarily. The correct constant can be estimated if we have *a priori* knowledge of the maximum acceleration of the particles.

than the cost when we do not approximate at all, while for high accuracy this optimal cost is higher than the original. This can also be explained by our model since the cost of the approximation increases, both in terms of the number of polynomial pieces required to approximate the original trajectory and the cost of the least squares fit, which, as we saw, depends on the imposed accuracy.

The lesson from these experiments is that, if we require very accurate approximate trajectories, then we are better off performing the simulation without doing the approximation and taking advantage of the speed-up provided by our interval approach. If accuracy is not an issue, then a smaller degree will be advantageous and the above analysis and experimental data offer some guidance on the choice of the optimal degree.

8 Conclusions

In this paper we have presented an interval-based method for maintaining kinetic simulations of objects that move on polynomial trajectories. The major idea of the method is to use intervals that contain the event times of the simulation in order to resolve the comparisons between events times in the event queue, thus avoiding wasting time on computing event times for events that may never occur, or computing them more accurately than needed. Experimental results, as well as a simple theoretical analysis, show that by using the interval-based method we gain a speed-up of d, where d is the degree of motion, over the naive approach.

Although polynomial motions constitute a common class of motions, we would like to extend our approach to more general motions. In particular, we would like to explore the possibility of applying our algorithm to motions that are solutions of ordinary differential equations either by exploiting existing theoretical results (see [7]) or by approximating the solution of the o.d.e. by a polynomial function and then adding additional certificates in the kinetic simulation corresponding to the times that the particular approximations are no longer valid. Another possible direction of research is to use interval arithmetic techniques to obtain bounds on function values (see [12]) for non-polynomial functions, and use these bounds as the basis for our approach.

9 Acknowledgments

The authors wish to thank Pankaj Agarwal, Julien Basch, Dan Halperin and Li Zhang for useful discussions.

References

[1] G. Alefeld and J. Herzberger. *Introduction to interval computations*. Academic Press, New York, 1983.

[2] J. Basch, L. J. Guibas, and J. Hershberger. Data structures for mobile data. In *Proc. 8th ACM-SIAM Symp. Discrete Algorithms*, pages 747–756, 1997.

[3] J. L. Bentley and T. Ottman. Algorithms for reporting and counting geometria intersections. *IEEE Trans. on Computers*, C-28(9):643–467, Sept. 1973.

[4] L. J. Guibas. Kinetic data structures — a state of the art report. In *Proc. 3rd Work. Algorithmic Found. of Robotics*, 1998. To appear.

[5] N. Jacobson. *Basic Algebra I*. W. H. Freeman, New York, 2nd edition, 1985.

[6] M. A. Jenkins. Algorithm 493 zeros of a real polynomial. *ACM Transactions on Mathematical Software*, 1:178–, 1975.

[7] A. G. Khovanskiĭ. *Fewnomials*, volume 88 of *Translations of Mathematical Monographs*. Americal Mathematical Society, Providence, Rhode Island, 1991.

[8] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[9] M. Lang and B.-C. Frenzel. Polynomial root finding. *IEEE Signal Processing Letters*, 1994.

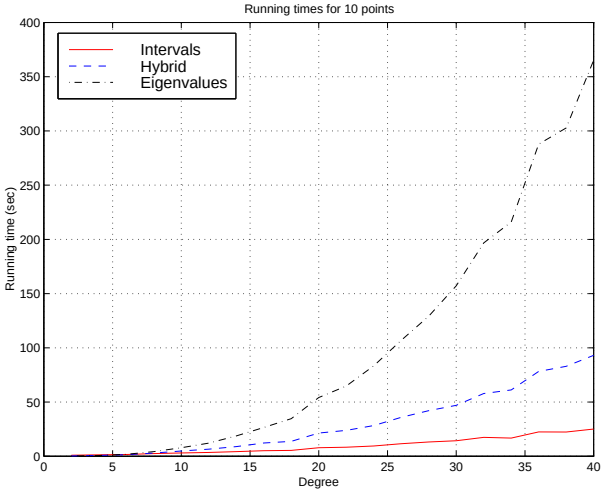
[10] J. Peetre. Approximation of norms. *J. Approx. Theory*, 3(3):243–260, 1970.

[11] W. H. Press, S. A. Teukolsky, W. T. Vettering, and B. P. Flannery. *Numerical Recipes in C*. Cambridge University Press, Cambridge, 2nd edition, 1992.

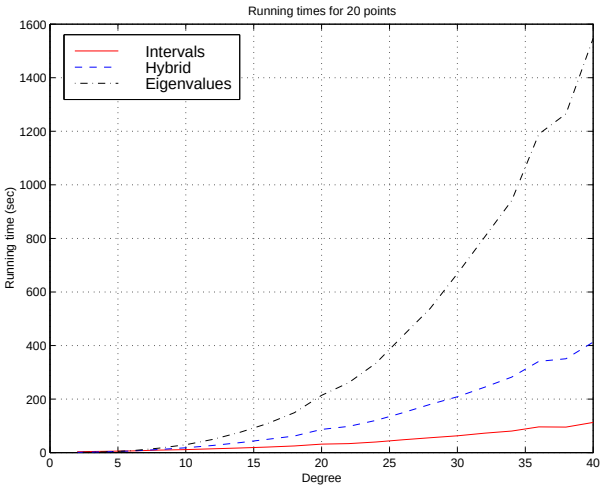
[12] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. John Wiley & Sons, New York, 1984.

[13] L. N. Trefethen and D. Bau, III. *Numerical Linear Algebra*. SIAM, 1997.

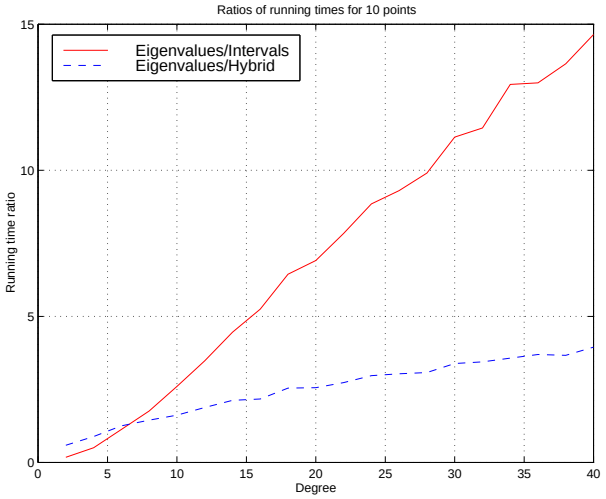
[14] R. E. Zippel. *Effective polynomial computation*. Kluwer Academic Publishers, Boston, 1993.



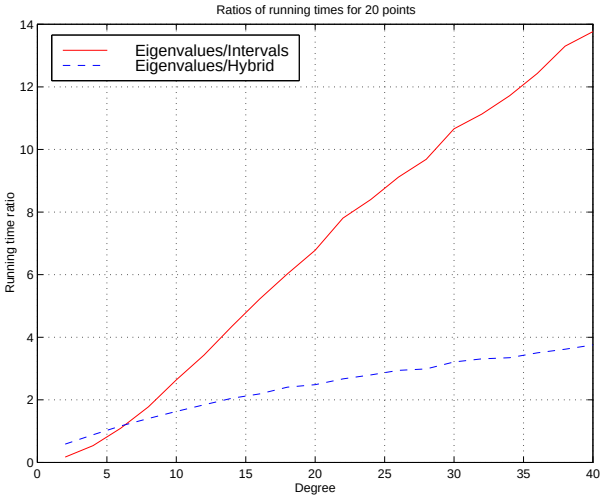
(a) DT: running times for 10 points



(b) DT: running times for 20 points

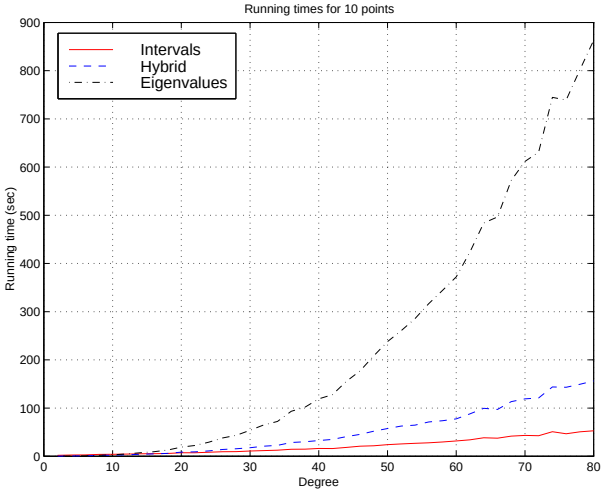


(c) DT: ratios of running times for 10 points

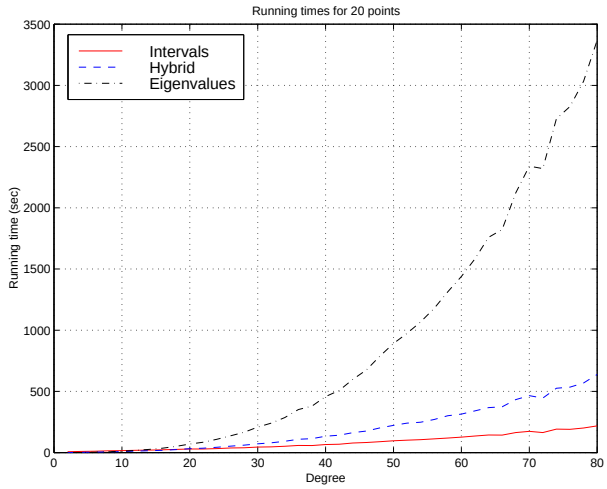


(d) DT: ratios of running times for 20 points

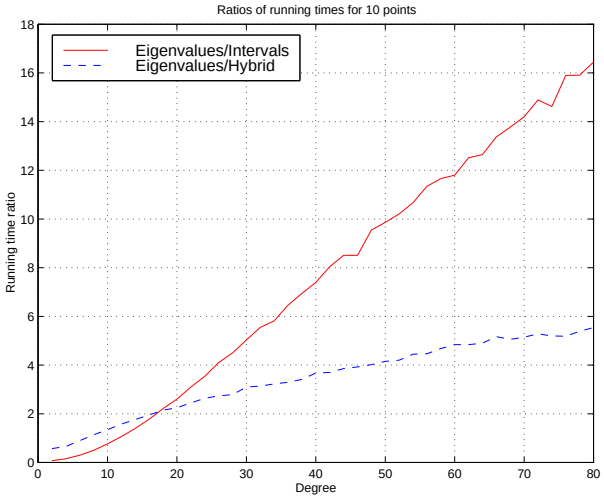
Figure 1: Mean running times in seconds and ratios of running times for maintaining the Delaunay triangulation of 10 and 20 points on a plane using the three different methods for handling the events times: the interval-based, the eigenvalue one and a hybrid one; 10 initial configurations were used for each point set.



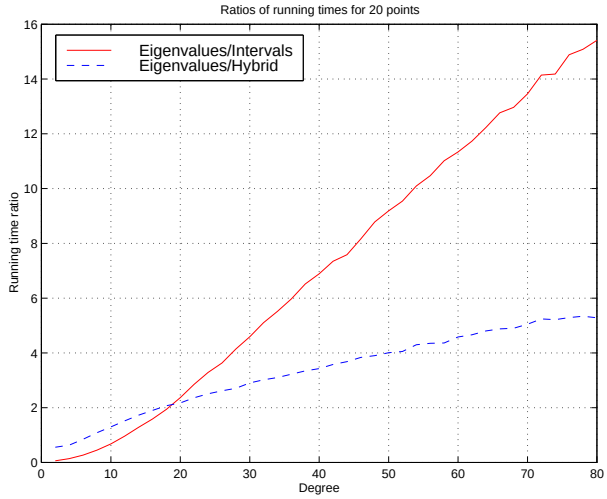
(a) CP: running times for 10 points



(b) CP: running times for 20 points



(c) CP: ratios of running times for 10 points



(d) CP: ratios of running times for 20 points

Figure 2: Mean running times in seconds and ratios of running times for maintaining the closest pair of 10 and 20 points on a plane using the three different methods for handling the events times: the interval-based, the eigenvalue one and a hybrid one; 10 initial configurations were used for each point set.

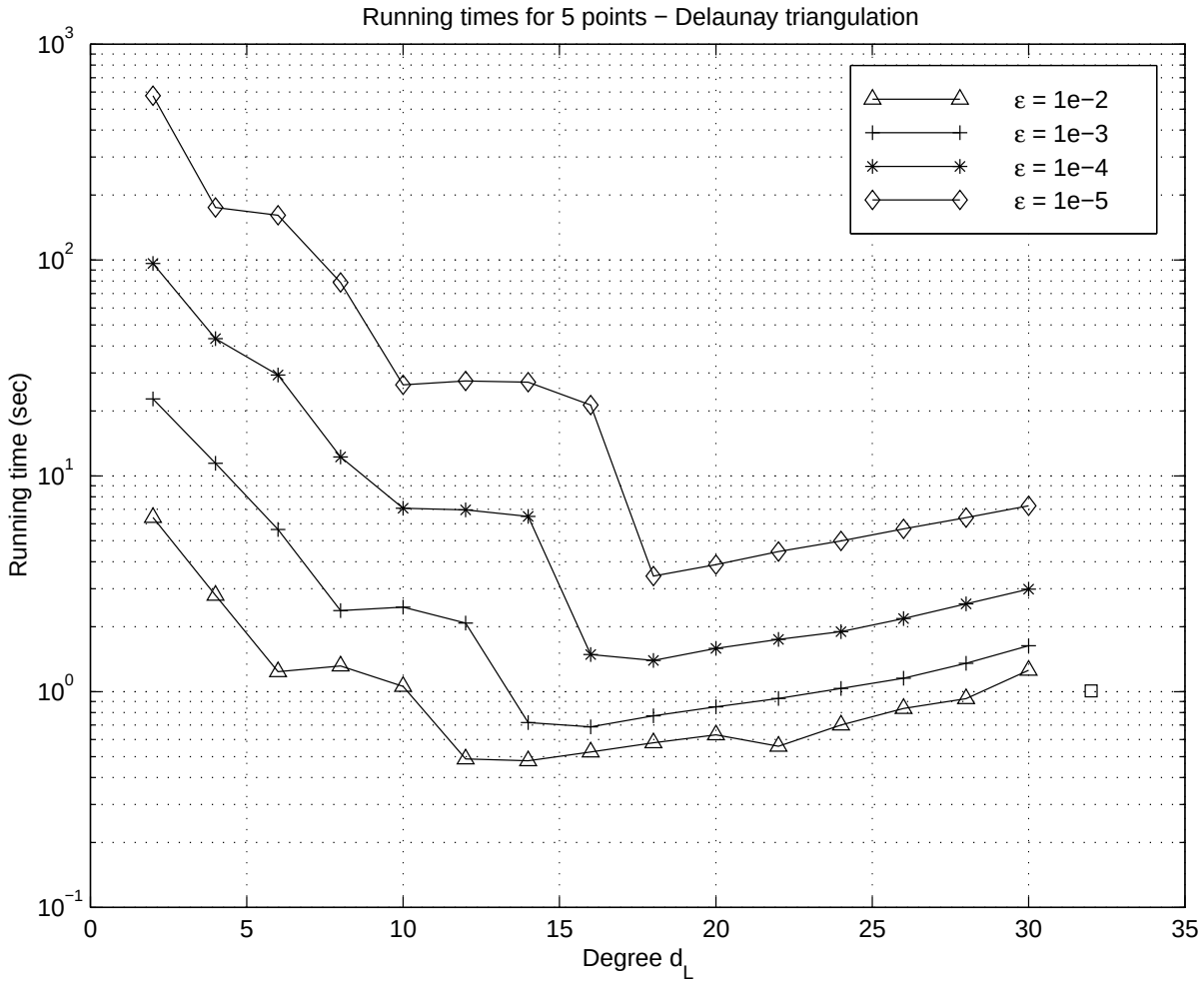


Figure 3: Mean running times in seconds for maintaining the Delaunay triangulation of 5 moving points as a function of the degree d_L of the approximate splined motions. The points are moving originally on polynomial trajectories of degree $d_H = 32$; the running time for the simulation using the original trajectory is shown by a square. Four different values for ϵ are considered: 10^{-i}, $i = 2, 3, 4, 5$. The stop time is $T_{max} = 1$. 10 initial configurations are used for each point set. The interval-based method is applied.