Fast High-Resolution Drawing of Algebraic Curves

Nuwan Herath Mudiyanselage, Guillaume Moroz, Marc Pouget

▶ **To cite this version:**

Nuwan Herath Mudiyanselage, Guillaume Moroz, Marc Pouget. Fast High-Resolution Drawing of Algebraic Curves. ISSAC 2022 - International Symposium on Symbolic and Algebraic Computation, Jul 2022, Villeneuve-d'Ascq France, France. pp.449-458, 10.1145/3476446.3535483 . hal-03788409

HAL Id: hal-03788409
https://inria.hal.science/hal-03788409v1

Submitted on 26 Sep 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Fast High-Resolution Drawing of Algebraic Curves

Nuwan Herath Mudiyanselage
nuwan.herath-mudiyanselage@inria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Guillaume Moroz
guillaume.moroz@inria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

Marc Pouget
marc.pouget@inria.fr
Université de Lorraine, CNRS, Inria, LORIA
Nancy, France

ABSTRACT

We address the problem of computing a drawing of high resolution of a plane curve defined by a bivariate polynomial equation $P(x,y)=0$. Given a grid of fixed resolution, a drawing is a subset of pixels. Our goal is to compute an approximate drawing that (i) contains all the parts of the curve that intersect the pixel edges, (ii) excludes a pixel when the evaluation of P with interval arithmetic on each of its four edges is far from zero.

One of the challenges for computing drawings on a high-resolution grid is to minimize the complexity due to the evaluation of the input polynomial. Most state-of-the-art approaches focus on bounding the number of independent evaluations. Using state-of-the-art Computer Algebra techniques, we design new algorithms that amortize the evaluations and improve the complexity for computing such drawings.

Our main contribution is to use a non-uniform grid based on the Chebyshev nodes to take advantage of multipoint evaluation techniques via the Discrete Cosine Transform. We propose two new algorithms that compute drawings and compare them experimentally on several classes of high degree polynomials. Notably, one of those approaches is faster than state-of-the-art drawing software.

CCS CONCEPTS

• **Theory of computation → Computational geometry**.

KEYWORDS

curve drawing, high degree algebraic curve, discrete cosine transform, numerical error analysis, fast multipoint evaluation

ACM Reference Format:
Nuwan Herath Mudiyanselage, Guillaume Moroz, and Marc Pouget. 2022. Fast High-Resolution Drawing of Algebraic Curves. In *Proceedings of the 47th ACM Symposium on Symbolic and Algebraic Computation.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

In several engineering applications such as mechanism design or control theory, it is important to visualize curves given by implicit equations of the form $P(x,y)=0$. Being able to draw them quickly with a good resolution is also an advantage in an interactive interface when the designer of a robotic mechanism wants to visualize different implicit curves associated to different design parameters.

In most state-of-the-art approaches computing the implicit curve solution of $P(x,y)=0$ [3, 32, 38], the authors analyze and optimize their algorithms while assuming that the cost of evaluating the function P is constant. On the other hand, in some applications, the function P can be a polynomial of degree 20 for example, with more than 200 terms, and evaluating it is not a negligible constant. When P is a polynomial, we will present two methods to speed up the computation of the drawing of the implicit curve $P(x,y)=0$, notably by amortizing the cost of the evaluations of P. Experimentally, we show that one of those methods computes the drawing an order of magnitude faster than state-of-the-art implicit-plot software. Moreover, our approach provides guarantees on the output drawing.

State of the art. The first efficient algorithm to visualize such an object was the marching cube [26], and was enhanced with variants such as surface nets [12] and dual contouring [20]. These algorithms are based on the evaluation of the function P on a regular grid. When the curve is smooth and the grid resolution is not fixed *a priori*, interval analysis can be used with a recursive subdivision of the input domain to guarantee a topologically correct reconstruction of the curve [3, 8, 10, 25, 32, 38]. Another approach [40] uses subdivision to exclude large parts of the domain and reduce the number of evaluations. When the resolution of the grid is fixed *a priori*, there is no hope to capture the topology of the curve but interval analysis can still be used to determine an enclosure: a set of pixels whose union contains the curve [40]. Another approach is the so-called predictor-corrector continuation that follows the curve by stepping in the tangent direction and correcting by projecting back to the curve via a Newton operator [13, Chapter 6]. The main problems are then to find starting points on the curve and make sure not to jump between different branches of the curve. For a singular curve, the topology can be computed when P is a polynomial, using symbolical approaches based notably on Cylindrical Algebraic Decomposition ([1, 2, 9, 11, 22] and references therein). The complexity of these methods is high with respect to the degree of P [22], and in practice it cannot handle random bivariate polynomials of degree 100. Since we work within a grid of fixed resolution and with a fixed precision, our approach does not compute the topology of the curve. We instead focus on reducing the cost of the evaluations of the polynomial P when the resolution is high and the precision is fixed.

Contribution. Our contribution focuses on the case where P is a polynomial of degree d and the input grid is of high resolution N. The main idea to speed up the implicit drawing computation is to take advantage of fast multipoint evaluation from computer algebra.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ISSAC '22, July 4–7, 2022, Villeneuve-d'Ascq, France.
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-8688-3/22/07…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

Indeed, a polynomial of degree d can be evaluated at one value in linear arithmetic complexity using the Horner algorithm. The naive evaluation at d values is thus quadratic in d. The fast multipoint evaluation algorithm improves the complexity of these d evaluations to soft-linear in d (that is $O(d\text{polylog}(d))$) [44, chapter 10]. On the other hand, fast multipoint evaluation algorithms have a bit complexity quadratic in d if we work with a fixed constant precision lower than d [23, 41]. If we restrict our multipoint evaluation on a set of Chebyshev nodes (see Section 2.1), we can then use the Discrete Cosine Transform (DCT), in the same way as the Discrete Fourier Transform (DFT) can be used for multipoint evaluation on the roots of unity in the complex field [44, §8.2]. In this case, the bit complexity is linear in d, even with a precision lower than d [36, §3]. This special multipoint evaluation inherits the numerical stability of the DCT and we extend the error analysis already known for the DFT [5].

We derive this idea with two algorithm variants. First, in Section 4.1, we design an algorithm that evaluates the input polynomial P on all nodes of a grid using the DCT both in x and in y. This constrains us to use a non-uniform grid aligned on Chebyshev nodes. Although this grid is non-uniform, the distance between two consecutive nodes of such a grid of size $N \times N$ is always less than π/N, which makes it sufficiently dense for plotting. Then, writing $P(X,Y) = \sum_{i=0}^{d} p_i(X)Y^j$, we can use the DCT on each p_i, which leads to N univariate polynomials $q_j(Y)$ of degree d, that can also be evaluated using the DCT again. This can be done in a quasi-quadratic number of arithmetic operations in N if $N \gg d$.

The second variant comes by mixing in an idea coming from interval arithmetic. For the computation of implicit curves, interval arithmetic combined with quad-tree subdivision approaches can discard large parts of the plane with few evaluations and reduce the number of evaluations required to plot the considered curve [32, 38]. In our case, we further improve this approach in Section 4.2 by first evaluating all the p_i using the DCT, and then we solve each polynomial $q_j(Y)$ with a subdivision approach based on interval arithmetic. If $N \gg d$, this leads to a complexity in $O(dNT)$, where $\log(N) < T < N$ is the maximum number of nodes in the considered subdivision trees. In Section 5, we show that this approach performs well in practice.

Finally, even though we use fast evaluation techniques for drawing implicit curves, the drawing returned by our algorithms are crossing-edge approximations as defined in Definition 4.3. In particular, we guarantee that we don't miss any intersection point of the curve with the underlying grid and that we exclude pixels when the evaluation of P on their edges is far from zero. Such guarantees are obtained using a combination of interval arithmetic and a careful analysis of the numerical error of the DCT algorithm in Section 3. One of the difficulties we encountered is that even though this analysis of the DCT allows us to bound the values of a polynomial on the Chebyshev nodes, it cannot bound its values on small intervals around the Chebyshev nodes. We solve this problem by using Taylor approximations of order m with a bound on the remainder, where m is a small constant, say 2 or 3. The Taylor approximations at all the Chebyshev nodes can be efficiently computed using the DCT algorithm m times.

We implemented our two approaches in Python. In Section 5, we remark that the timings match the complexity analysis. We also compare our implementations with state-of-the-art software, and show that for high resolutions, the approach based on a mixed strategy using fast multipoint evaluations and subdivision proves to be the fastest.

2 FAST MULTIPOINT EVALUATION OF POLYNOMIALS ON CHEBYSHEV NODES

We first recall the definitions of Chebyshev polynomials and nodes, and the Inverse Discrete Cosine Transform (IDCT). We then show that the IDCT enables a fast multipoint evaluation of a polynomial at the Chebyshev nodes. Section 2.3 introduces notation for interval arithmetic.

2.1 Chebyshev nodes, Chebyshev basis and IDCT

Chebyshev polynomials can be defined as the unique sequence of polynomials $(T_n)_{n\in\mathbb{N}}$ satisfying

$$T_n(\cos\theta) = \cos(n\theta). \tag{1}$$

For $N \in \mathbb{N}$, the Chebyshev nodes $(c_k)_{k\in[\![0,N-1]\!]}$ are the roots of T_N:

$$c_k = \cos\left(\frac{2k+1}{2N}\pi\right) \text{ for } k \in [\![0,N-1]\!]. \tag{2}$$

The Chebyshev polynomials T_k for $k \le d$ form a basis of the polynomials of degree at most d. The relation between the monomial and the Chebyshev bases is given by [28, section 2.3.1]:

$$x^k = 2^{1-k} {\sum_{i=0}^{\lfloor k/2 \rfloor}}' \binom{k}{i} T_{k-2i}(x),$$

where the prime symbol denotes that the term $T_0(x)$, if there is one, is to be halved.

Lemma 2.1. *For a polynomial of degree d, the change-of-basis matrix from the monomial basis to the Chebyshev basis is $B = (B_{i,j}) \in \mathbb{R}^{(d+1)\times(d+1)}$ where*

$$B_{i,j} = \begin{cases} 2^{-j}\binom{j}{\frac{j}{2}} & \text{if } i = 0 \text{ and } j \text{ is even,} \\ 2^{1-j}\binom{j}{\frac{j-i}{2}} & \text{if } \exists k, i = j-2k \text{ and } i \le j, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Performing a change of basis is a multiplication of a $(d+1)\times(d+1)$-matrix by a $(d+1)$-vector. Thus, given a polynomial of degree d in the monomial basis, computing its coefficients in the Chebyshev basis can be done in $O(d^2 p \log(p))$ bit-operations in precision-p arithmetic. Remark that it is also possible to do the change of basis in $O(d\ \text{polylog}(d))$ arithmetic operations [4, 31], however those methods are based on Taylor shift operations that requires a number of bit operations quadratic in d [43].

Definition 2.2. Given $d, N \in \mathbb{N}^*$ with $d+1 \le N$ and $(X_i)_{i\in[\![0,d]\!]}$ a sequence of real numbers, the Inverse Discrete Cosine Transform $\text{IDCT}((X_i)_{i\in[\![0,d]\!]}, N)$ is the sequence $(x_k)_{k\in[\![0,N-1]\!]}$ defined by

$$\forall k \in [\![0,N-1]\!], x_k = \frac{1}{N}\left(\frac{1}{2}X_0 + \sum_{i=1}^{d} X_i \cos\left(\frac{i(2k+1)}{2N}\pi\right)\right). \tag{4}$$

When input and output sizes match, it is omitted: $\text{IDCT}((X_i)_{i\in[\![0,d]\!]})$ denotes $\text{IDCT}((X_i)_{i\in[\![0,d]\!]}, d+1)$.

2.2 Fast multipoint evaluation FME

Let P be a polynomial of degree d given in the Chebyshev basis $P(x)=\sum_{i=0}^{d} a_i T_i(x)$. Using Eq. (1), the evaluation of P at a Chebychev node c_k of T_N for $N>d$ satisfies $\forall k \in [\![0,N-1]\!]$

$$P(c_k)=\sum_{i=0}^{d} a_i T_i\left(\cos\left(\frac{2k+1}{2N}\pi\right)\right)=\sum_{i=0}^{d} a_i \cos\left(\frac{i(2k+1)}{2N}\pi\right)$$
$$=\frac{a_0}{2}+\left[\frac{a_0}{2}+\sum_{i=1}^{d} a_i \cos\left(\frac{i(2k+1)}{2N}\pi\right)\right].$$

All the evaluations can thus be expressed using the IDCT as

$$(P(c_i))_{i\in[\![0,N-1]\!]}=N\cdot \mathrm{IDCT}((a_i)_{i\in[\![0,d]\!]},N)+\frac{1}{2}(a_0,\ldots,a_0). \quad (5)$$

When the polynomial P is given in the monomial basis, one has to first perform a change of basis to take advantage of the multipoint evaluation via the IDCT. The Fast Multipoint Evaluation operator FME is the composition of these operations.

Definition 2.3. For a polynomial $P=\sum_{i=0}^{d}\alpha_i x^i$, let us define $\mathrm{FME}((\alpha_i)_{i\in[\![0,d]\!]},N)$ the Fast Multipoint Evaluation of P at the Chebyshev nodes $(c_k)_{k\in[\![0,N-1]\!]}$ computed by a change of basis and Eq. (5).

Using the Fast Fourier Transform, the complexity of the IDCT is $O(N\log_2(N))$. Together with Lemma 2.1, this gives the complexity of the FME.

Lemma 2.4. *The computation of the* FME *has complexity* $O(d^2+N\log_2(N))$.

2.3 Interval arithmetic

We use interval representations for all data in our algorithms and use interval arithmetic for the computations. We denote the set of intervals of $\mathbb{R}$ by $\mathbb{IR}$. Given a function $f:\mathbb{R}\to\mathbb{R}$, we call inclusion of f, a function $\square f:\mathbb{IR}\to\mathbb{IR}$ such that the set $f(I)=\{f(x)|x\in I\}$ is contained in $\square f(I)$, for all $I\in\mathbb{IR}$. These definitions naturally extend to the multivariate setting and we refer to [29] for details. In particular, the arithmetic operations have natural extensions to intervals. Using such interval operations to evaluate a polynomial P with a given scheme gives an inclusion function $\square P$ for P. Several techniques exist to enclose the evaluation of a function on an interval, such as the Horner scheme, the compensated Horner scheme, the centered form, the extended Horner scheme ([30, Chapter 2], [39, Chapter 3], [14]), or more recently using recursive Lagrange interpolation when the evaluations are done within a dichotomic algorithm [17]. In our work, for a given integer m, we use the Taylor form of order m ([33, Definition 3.3], [17]) and we recall the error bounds based on Taylor-Lagrange inequality in Section 3.4.

3 NUMERICAL ERROR BOUNDS

In this section, we derive numerical error bounds for the fast multipoint evaluation used in Algorithms 1 and 3, and for the Taylor approximation used in Algorithm 1. In Section 3.1, we recall how the IDCT is computed via the Inverse Discrete Fourier Transform (IDFT). In Section 3.1.3, we recall previous work on the IFFT error where the IFFT is an algorithm computing the IDFT in quasi-linear number of operations. In Section 3.1.4, we derive an error bound for the IDCT when performing operations in precision-p arithmetic. Analysing

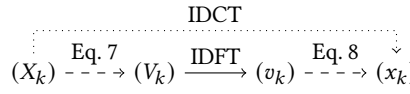


Figure 1: Fast IDCT procedure with an IDFT on N real points

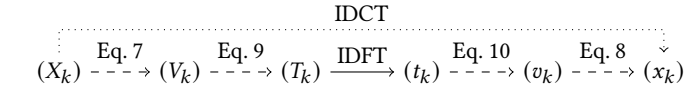


Figure 2: Fast IDCT procedure with an IDFT on $N/2$ complex points.

the numerical error due to the change from the monomial basis to the Chebyshev basis, we obtain the numerical error for the FME (Section 3.2) and its interval version $\square$FME (Section 3.3). In Section 3.4, we bound the error produced by using a low degree Taylor approximation of a high degree polynomial that is instrumental in Algorithm 1.

We assume without loss of generality that $N=2^n$ is a power of 2. Let j be such that $j^2=-1$. The IDFT is defined for a complex vector $z=(z_k)_{k\in[\![0,N-1]\!]}$ by

$$IDFT(z)=\left(\frac{1}{N}\sum_{i=0}^{N-1} z_k e^{j\frac{2\pi}{N}ik}\right)_{k\in[\![0,N-1]\!]} \quad (6)$$

3.1 Fast IDCT error bound

3.1.1 Reduction of the IDCT to N-points IDFT. This section recalls how the fast IDCT is computed by Makhoul [27]. The goal is to reduce the computation of (x_k), the IDCT of (X_k), to the computation of (v_k), the IDFT of (V_k)(Figure 1).

Let $\omega_M=e^{-j2\pi/M}$. Given $(X_k)_{k\in[\![0,N-1]\!]}$ and $X_N=0$, (V_k) is defined by

$$V_k=\frac{1}{2}\omega_{4N}^{-k}[X_k-jX_{N-k}],\ 0\le k\le N-1, \quad (7)$$

and (x_k) is retrieved from

$$\begin{cases} x_{2k}=v_k, & 0\le k\le \lfloor\frac{N-1}{2}\rfloor,\\ x_{2k+1}=v_{N-k-1}, & 0\le k\le \lfloor\frac{N}{2}\rfloor-1. \end{cases} \quad (8)$$

3.1.2 Reduction of the IDCT to $(N/2)$-points IDFT. We can further reduce the size of the sequence computed through IDFT. Since (V_k) is a Hermitian symmetric sequence, the number of points for the IDFT can be divided by 2 (Figure 2).

(T_k) is computed following the flow graph in Figure 3, for $0\le k\le \lfloor N/4\rfloor$, where $\overline{\bullet}$ denotes the complex conjugate (see Figure 3):

$$\begin{aligned} T_k&=\frac{1}{2}\left[\left(V_k+\overline{V_{\frac{N}{2}-k}}\right)+j\omega_N^{-k}\left(V_k-\overline{V_{\frac{N}{2}-k}}\right)\right],\\ \overline{T_{\frac{N}{2}-k}}&=\frac{1}{2}\left[\left(V_k+\overline{V_{\frac{N}{2}-k}}\right)-j\omega_N^{-k}\left(V_k-\overline{V_{\frac{N}{2}-k}}\right)\right]. \end{aligned} \quad (9)$$

The (N/2)-point IDFT of (T_k) gives (t_k). The sequence (v_k) is obtained thanks to

$$\begin{aligned} v_{2k}&=Re(t_k)\\ v_{2k+1}&=Im(t_k) \end{aligned} \quad (10)$$

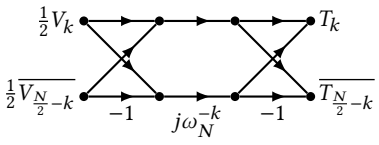


Figure 3: Flow graph to compute (T_k) from (V_k).

3.1.3 IFFT error bound. Based on the error bound given in Brisebarre et al. [5, Theorem 3.3 and 3.4] on the FFT, we can write a bound on the Inverse FFT in Corollary 3.1.

Corollary 3.1. *Assume radix-2, precision-p arithmetic, with rounding unit $u=2^{-p}$. Let $\widehat{z}$ be the computed 2^n-point IFFT of $Z\in\mathbb{C}^{2^n}$ and let z be the exact value. Then*

$$\|\widehat{z}-z\|_2 \le \|z\|_2\left[(1+u)^n(1+g)^{n-2}-1\right]$$

with

$$g=\frac{\sqrt{2}}{2}u+\rho_\times\left(1+\frac{\sqrt{2}}{2}u\right)$$

$$\rho_\times=\begin{cases} u\sqrt{5} & \text{naive multiplication,}\\ 2u & \text{multiplication with fused multiply-add instruction.}\end{cases}$$

For our application, we actually want to bound $\|\widehat{z}-z\|_\infty$. Using classical results on the equivalence between norms, and the equality $\|Z\|_2=\sqrt{N}\|z\|_2$, we deduce Corollary 3.2, where

$$\|Z\|_\infty^\perp = \max_{i\in[\![0,N-1]\!]}\{\max(|Re(Z_i)|,|Im(Z_i)|)\}.$$

Corollary 3.2. *Assume radix-2, precision-p arithmetic, with rounding unit $u=2^{-p}$. Let $\widehat{z}$ be then computed 2^n-point IFFT of $Z\in\mathbb{C}^{2^n}$ and let z be the exact value. Then*

$$\|\widehat{z}-z\|_\infty^\perp \le \|Z\|_\infty^\perp\sqrt{2}\left[(1+u)^n(1+g)^{n-2}-1\right].$$

3.1.4 Fast IDCT error bound. Using the notation of Section 3.1, Theorem 3.3 shows that the absolute error $\|\widehat{x}-x\|_\infty$ on the output of the fast IDCT can be bounded with respect to u, g (defined in Corollary 3.1) and $\|X\|_\infty$.

Theorem 3.3. *Assume radix-2, precision-p arithmetic, with rounding unit $u=2^{-p}$. Let $\widehat{x}$ be the computed 2^n-point fast IDCT of $X\in\mathbb{C}^{2^n}$ and let x be the exact value. Then*

$$\begin{aligned}&\|\widehat{x}-x\|_\infty\\ &\quad\le\sqrt{2}\|X\|_\infty\Big[\sqrt{2}(1+u)^3(1+g)^2\Big((1+u)^{n-1}(1+g)^{n-3}-1\Big)\\ &\qquad\qquad+(1+u)^3(1+g)^2-1\Big].\end{aligned}$$

Proof. The computation of the IDCT corresponds to the operations from X_k to x_k in Table 1 where the relative error for the norm $\|\cdot\|_\infty^\perp$ is bounded for each step in the last column.

Let $\widehat{T}$ (resp. $\widehat{t}$) be the computed value of the first two (resp. three) steps in Table 1, assuming precision-p arithmetic. Let us define $t^*=IDFT(\widehat{T})$ and $t=IDFT(T)$ the exact values. Taking into account the relative error at each step and using Cor. 3.2, we have

$$\begin{aligned}\|\widehat{x}-x\|_\infty &= \left\|\widehat{t}-t\right\|_\infty^\perp \le \left\|\widehat{t}-t^*\right\|_\infty^\perp+\left\|t^*-t\right\|_\infty^\perp\\ &\le \left\|\widehat{T}\right\|_\infty\sqrt{2}\Big((1+u)^{n-1}(1+g)^{n-3}-1\Big)+\left\|\widehat{T}-T\right\|_\infty.\end{aligned}$$

Moreover,

$$\left\|\widehat{T}\right\|_\infty \le \|T\|_\infty(1+u)^3(1+g)^2$$

and

$$\left\|\widehat{T}-T\right\|_\infty \le \|T\|_\infty((1+u)^3(1+g)^2-1).$$

We can also prove that $\|T\|_\infty\le\sqrt{2}\|X\|_\infty$, so

$$\begin{aligned}&\|\widehat{x}-x\|_\infty\\ &\quad\le\sqrt{2}\|X\|_\infty\Big[\sqrt{2}(1+u)^3(1+g)^2\Big((1+u)^{n-1}(1+g)^{n-3}-1\Big)\\ &\qquad\qquad+(1+u)^3(1+g)^2-1\Big].\end{aligned}$$

□

The ratio $\|\widehat{x}-x\|_\infty/\|X\|_\infty$ is computed for high resolutions in Table 2 and it does not exceed 10^{-13}.

3.2 FME error bound

For the fast multipoint evaluation, the polynomial is written in the Chebyshev basis and then the IDCT is applied on these coefficients. The change of basis introduces an error which has to be added to the error from the IDCT to get a bound for the FME in Theorem 3.4.

Theorem 3.4. *Assume radix-2, precision-p arithmetic, with rounding unit $u=2^{-p}$. Let $\widehat{z}$ be the computed 2^n-point fast evaluation on Chebyshev nodes of the polynomial whose coefficients are $a\in\mathbb{R}^{d+1}$ and let z be the exact value. Then*

$$\|z-\widehat{z}\|_\infty \le (d+1)\|a\|_\infty\left[(d+1)\gamma+(1+\gamma)\left(N\beta(1+u)+\left(d+\frac{3}{2}\right)u\right)\right]$$

where

$$\begin{aligned}\gamma&=\frac{(d+1)u}{1-(d+1)u},\\ \beta&=\sqrt{2}\Big[\sqrt{2}(1+u)^3(1+g)^2\Big((1+u)^{n-1}(1+g)^{n-3}-1\Big)\\ &\quad+(1+u)^3(1+g)^2-1\Big].\end{aligned}$$

Proof. The FME computation consists essentially in writing the input polynomial in the Chebyshev basis using Eq. (3), and then computing an IDCT. For the error bound on the change of basis, we use a classical bound on the inner product [16, §3.1]. For the IDCT we use the bound in Theorem 3.3. □

3.3 FME with interval coefficients

In the case where the input polynomial has interval coefficients, we need to compute the interval enclosure of its evaluation on the Chebyshev nodes. This leads to a function denoted by □FME that takes as input a polynomial with interval coefficients, and returns a list of intervals that each contain the evaluation of the input polynomial on a Chebyshev node. The function □FME is computed in two steps. First we compute the change of basis through a multiplication with the matrix given in Equation 3. Then we compute the IDCT on

Table 1: Summary of the operations for the IDCT and their relative errors.

	Operations	Floating point relative errors
$X_k \to V_k$	$V_k = \frac{1}{2}\omega_N^{-k}(X_k - jX_{N-k})$	g (since X_k real)
$V_k \to T_k$	$T_k = \frac{1}{2}((V_k + \overline{V_{\frac{N}{2}-k}}) + j\omega_N^{-k}(V_k - \overline{V_{\frac{N}{2}-k}}))$ $T_{\frac{N}{2}-k} = \frac{1}{2}((V_k + \overline{V_{\frac{N}{2}-k}}) - j\omega_N^{-k}(V_k - \overline{V_{\frac{N}{2}-k}}))$	$(1+g)(1+u)^3 - 1$
$T_k \to t_k$	$t_k = IFFT(T)_k$	$\sqrt{2}((1+u)^{n-1}(1+g)^{n-3} - 1)$ Cor. 3.2 for $\frac{N}{2} = 2^{n-1}$ points
$t_k \to v_k$	$v_{2k} = Re(t_k)$ $v_{2k+1} = Im(t_k)$	0
$v_k \to x_k$	$x_{2k} = v_k$ $x_{2k+1} = v_{N-k-1}$	0

N	1024	2048	4096	8192	16384	32768
$\|\widehat{x}-x\|_\infty / \|X\|_\infty$	7.97e-15	8.84e-15	9.72e-15	1.06e-14	1.15e-14	1.23e-14

Table 2: IDCT error bounds for $p = 53$ (double precision) using Theorem 3.3.

a vector of intervals. A challenge is to keep a tight inclusion and a complexity quasi-linear in N.

Definition 3.5. Let $A \in \mathbb{IR}^{d+1}$ be the interval coefficients of a polynomial of degree d and let $X = (X_0,\ldots,X_d) = BA$ where B is defined in Equation (3). Let $x \in \mathbb{R}^{d+1}$ be the vector of the centers of the intervals of X and r be the maximum of the radii of these intervals. Let $\widehat{x}$ be the result of the fast *IDCT* computation applied on x, and e be the bound on the error given in Theorem 3.3. We also let $E \in \mathbb{IR}^N$ be a vector where all the entries are $[-e - \frac{d+1}{N}r, e + \frac{d+1}{N}r]$. Finally, we define

$$\square\text{FME}(A) = N \cdot (\widehat{x} + E) + \frac{1}{2}(X_0,\ldots,X_0).$$

As a corollary of Theorem 3.3 on the error bound on the IDCT, we deduce a bound on the error of the FME function.

Corollary 3.6. *The function* □FME *is an inclusion of the function* FME *and requires* $O(N\log_2(N) + d^2)$ *arithmetic operations.*

Proof. First, let $Y \in \mathbb{IR}^N$ be the vector of the interval ranges IDCT(X). Using interval arithmetic, we have FME$(A) \subset N \cdot Y + \frac{1}{2}(X_0,\ldots,X_0)$ by definition of the FME operator in Section 2.2. Then, since the IDCT is linear, we have IDCT(X) = IDCT(x) + IDCT$(X - x)$. From Theorem 3.3, we have IDCT$(x) \subset \widehat{x} + e$. And since all the entries of $X - x$ are bounded by r, using the explicit formula for the IDCT given in Definition 2.2, we bound the absolute value of each entry of IDCT$(X - x)$ by $\frac{d+1}{N}r$. With the notation of Definition 3.5, this implies IDCT(x) + IDCT$(X - x) \subset \widehat{x} + E$, which concludes the proof for the bound. For the complexity, the dominating parts are the computation of the fast IDCT in $O(N\log_2(N))$ arithmetic operations and the change of basis in $O(d^2)$ operations. □

3.4 Bound with Taylor approximation

The Taylor-Lagrange inequality states that for a function f and two reals $a, b \in I$,

$$\left| f(b) - \sum_{k=0}^{m} \frac{1}{k!}(b-a)^k f^{(k)}(a) \right| \le \max_I \left| f^{(m+1)} \right| \frac{|b-a|^{m+1}}{(m+1)!}.$$

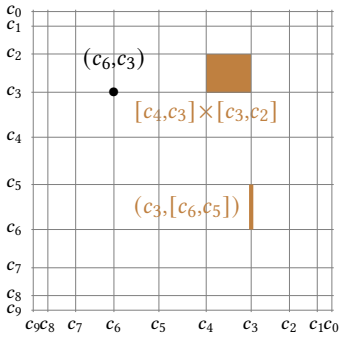


Figure 4: Chebyshev grid for $N = 10$, vertical segment $(c_3, [c_6, c_5])$ and pixel $[c_4, c_3] \times [c_3, c_2]$.

In Algorithm 1, the derivatives are evaluated using the □FME operator at the Chebyshev nodes. It then remains to use the Taylor-Lagrange inequality to bound the values in a neighborhood of each Chebyshev node. Theorem 3.7 provides two bounds, the second one is better for points close to -1 or 1.

Theorem 3.7. *For a polynomial* $P = \sum_{i=0}^{d} a_i X^i$, $c \in [-1, 1]$ *and* $r \in [-R, R]$

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \le \max_{i \in [\![0,d]\!]} |a_i| R^{m+1} \binom{d+1}{m+2}.$$

and if $|c \pm R| < 1$

$$\left| P(c+r) - \sum_{k=0}^{m} \frac{1}{k!} r^k P^{(k)}(c) \right| \le \max_{i \in [\![0,d]\!]} |a_i| \frac{R^{m+1}}{(1 - |c| - R)^{m+2}}$$

4 FAST CURVE ENCLOSURE

This section details our two algorithms for computing drawings of a polynomial curve $P(X, Y) = 0$. Both algorithms take advantage of the fast multipoint evaluation (□FME of Definition 3.5) with guaranteed error to partially evaluate the polynomial $P(X, Y)$ with respect to the X variable. Then the fibers $X = c_i$, that is the vertical lines, are processed either using the fast multipoint evaluation again together with a Taylor approximation (Algorithm 1 in Section 4.1), or using a classical subdivision algorithm (Algorithm 3 in Section 4.2). The input of our algorithms is a bivariate polynomial given in the monomial basis $P(X, Y) = \sum_{i=0}^{d} \sum_{j=0}^{d} a_{i,j} X^i Y^j$, and a Chebyshev grid of

size $N \times N$ illustrated in Figure 4. Note that the Chebyshev nodes (Eq. (2)) are naturally indexed in decreasing order, so that the domain covered by the grid is the square $[c_{N-1}, c_0]^2 \subsetneq [-1,1]^2$. To describe the output, we define segments on the Chebyshev grid.

Definition 4.1. A *vertical*, resp. *horizontal*, segment is defined by a scalar and an interval, resp. by an interval and a scalar. For instance, Figure 4 displays the vertical segment $(c_3, [c_6, c_5])$.

Definition 4.2. Let $E = 6(d+1)e(\|P\|_\infty, d, N, p)$, with $e(\|P\|_\infty, d, N, p)$ the bound given in Theorem 3.4. A segment S between two consecutive nodes of the grid is called:

- *crossed segment* if the curves intersects S,
- *candidate segment* if $0 \in \square P(S) + [-E, E]$.

Definition 4.3. A set of pixels is a *crossing-edge approximation* if the set of edges of its pixels contains all the crossed segments and is contained in the set of candidate segments.

1-pass version algorithms. The output of Algorithms 1 and 3 is a set of vertical segments with end points on the grid, that contains all the vertical crossed segments, and is contained in the set of vertical candidate segments, according to Lemma 4.4 and 4.6

2-pass version algorithms. We define the *2-pass versions* of Algorithm 1 or 3 as the procedure running this algorithm as originally specified and then running it again switching the roles of X and Y. One then obtains vertical and horizontal segments enclosing the intersections between the curve and all the lines of the grid. We call a *pixel* of the grid a rectangle defined by successive Chebyshev nodes, that is of the form $[c_{i+1}, c_i] \times [c_{j+1}, c_j]$ for $i, j \in \{0, \dots, N-2\}$, see Figure 4. The output of the 2-pass algorithms is the set of pixels that have (at least) a side covered by the above-mentioned set of vertical and horizontal segments. Our drawing thus is a crossing-edge approximation defined in Definition 4.3 that ensures that the 2-pass versions only miss small parts of the curve included in the interior of a pixel.

4.1 Multipoint partial evaluation and fast Taylor approximation

Algorithm 1 first uses the □FME (Definition 3.5) to partially evaluate the polynomial $P(X,Y)$ with respect to the X variable at all Chebyshev nodes (c_i) (for loop of Line 3). The resulting univariate interval polynomials $P(c_i, Y)$ take the error generated by the finite precision arithmetic into account. Each vertical fiber $X = c_i$ is then processed separately. The univariate polynomial $P(c_i, Y)$ and its derivatives up to order m are evaluated at all Chebyshev nodes (c_j) using again the □FME operator (for loop of Line 9). These data thus define a Taylor approximation of $P(c_i, Y)$ in a vertical neighborhood of each Chebyshev node c_j (Line 15). Finally, a bound on the values of P in this neighborhood is computed by the interval evaluation of this Taylor approximation together with Theorem 3.7 (Lines 16 to 18). When this interval evaluation contains the value 0, the algorithm outputs a vertical segment that may contain a true 0 value of P, that is an intersection with the curve $P(X,Y) = 0$ (Line 19).

Lemma 4.4. *The set of vertical segments returned by Algorithm 1 contains all the vertical crossed segments, and is contained in the set of candidate segments.*

Proof. The correctness of Algorithm 1 follows from the error analysis in Theorem 3.4 of the FME operator and the use of interval arithmetic in all the computations. □

Theorem 4.5. *The number of operations of Algorithm 1 is* $O(Nd^2 + N^2 \log_2(N) + d^3)$.

Proof. The partial evaluation of $P(X,Y) = \sum_{j=0}^{d} (\sum_{i=0}^{d} a_{i,j} X^i) Y^j$ in X is the evaluation of the $d+1$ polynomials $\sum_{i=0}^{d} a_{i,j} X^i$. In Line 4, each of these polynomials is evaluated via the □FME at all the Chebyshev nodes. The cost is $d+1$ times the cost of one □FME of size N, that is $O(d(d^2 + N\log_2(N)))$ according to Lemma 2.4. In the for loop of Line 9, for each of the N vertical fibers, the data for the degree m Taylor approximations are computed at all Chebyshev nodes using the □FME again for a complexity of $O(Nm(d^2 + N\log_2(N))) = O(N(d^2 + N\log_2(N)))$ assuming m constant. In Line 13, the computation of the maximum absolute value of the coefficients of $P(c_i, Y)$ is in $O(d)$ and thus $O(dN)$ for all fibers. In the for loop of Line 14, for each c_i and each c_j, the degree m Taylor approximation is evaluated by interval arithmetic on the corresponding vertical neighborhood and the error from Theorem 3.7 is added. All these operations are linear in m, this gives a total complexity of $O(mN^2) = O(N^2)$. The total complexity of Algorithm 1 is thus $O(Nd^2 + N^2\log_2(N) + d^3)$ since we assume $d < N$. □

4.2 Multipoint partial evaluation and subdivision

The first part of Algorithm 3 is the same as Algorithm 1, □FME is used for partial evaluations. In each vertical fiber $X = c_i$, we need to identify the vertical segments that enclose the intersection of the curve $P(X,Y) = 0$ with the fiber. This is done using Algorithm 2, which is one variant of the different root isolation algorithms in the literature that can handle polynomials with interval coefficients [6, 7, 21, 34, 35].

Lemma 4.6. *The set of vertical segments returned by Algorithm 3 contains all the vertical crossed segments, and is contained in the set of candidate segments.*

Proof. The correctness of Algorithm 3 follows from the error analysis in Theorem 3.4 of the FME operator and the use of interval arithmetic in all the computations. □

The complexity of the subdivision in Algorithm 2 depends on the size of its subdivision tree, whose depth is bounded in our variant by $\log_2(N)$. We are thus aiming at a complexity for Algorithm 3 that is output sensitive in T, the total number of nodes for the largest subdivision tree during the execution of the algorithm, that is over all the vertical fibers. In practice, one can expect that a curve crosses each fiber a constant number of times. If there is not many false positive segments in the output, $T = O(\log_2(N))$. In the worst case, one may have $T = O(N)$.

Theorem 4.7. *The number of operations of Algorithm 3 is* $O(d^3 + dN\log_2(N) + dNT)$, *where* T *is the maximum number of nodes of the subdivision trees over all vertical fibers.*

Proof. As in the proof of Theorem 4.5, the partial evaluations of for loop of Line 3) has complexity $O(d^3 + dN\log_2(N))$. In Line 8, in each vertical fiber, the subdivision Algorithm 2 performs $O(T)$ interval evaluations of the partially evaluated univariate polynomial

Algorithm 1 Multipoint partial evaluation with Taylor approximation

Input: A bivariate polynomial $P(X, Y) = \sum_{i,j} a_{i,j} X^i Y^j$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$, a Chebyshev grid resolution integer $N > 4d > 0$ and the order m of the Taylor approximation.

Output: A set of vertical segments containing all the vertical crossed segments and contained in the set of candidate segments (Definition 4.2)

procedure TAYLOR(P, N, m)
  $\mathbf{d} \in \mathbb{R}^{N\times(d+1)}$
  for $j \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{j=0}^{d} d_{i,j} Y^j = P(c_i, Y)$
    $d_{0,j},\ldots,d_{N-1,j} \leftarrow \square\text{FME}((a_{0,j},\ldots,a_{d,j}), N)$ ▷ N-point evaluation (Def. 3.5)
  end for
  $\mathcal{S} \leftarrow \emptyset$
  $\mathbf{q} \in \mathbb{R}^{d+1}, \mathbf{p} \in \mathbb{R}^{(m+1)\times N}$
  for $i \leftarrow 0$ **to** $N-1$ **do** ▷ Processing vertical fiber $X = c_i$
    for $k \leftarrow 0$ **to** m **do**
      $\sum_{j=0}^{d-k} q_j Y^j \leftarrow \text{diff}(\sum_{j=0}^{d} d_{i,j} Y^j, k)$ ▷ $\sum_{j=0}^{d-k} q_j Y^j = \frac{\partial^k}{\partial_Y^k} P(c_i, Y)$
      $p_{k,0},\ldots,p_{k,N-1} \leftarrow \square\text{FME}((q_0,\ldots,q_{d-k}), N)$ ▷ $p_{k,j} = \frac{\partial^k}{\partial_Y^k} P(c_i, c_j)$
    end for
    $d_{i,max} \leftarrow \max_{0\le j\le d} |d_{i,j}|$
    for $j \leftarrow 0$ **to** $N-1$ **do**
      $T \leftarrow \sum_{l=0}^{m} \frac{p_{l,j}}{l!} Y^l$ ▷ Degree m Taylor approximation at (c_i, c_j)
      if $j < N/2$ **then** $R \leftarrow \frac{c_j - c_{j+1}}{2}$ **else** $R \leftarrow \frac{c_{j-1} - c_j}{2}$ **end if**
      $\beta \leftarrow d_{i,max} R^{m+1} \min\left\{\frac{1}{(1-|c_j|-R)^{m+2}}, \binom{d+1}{m+2}\right\}$ ▷ see Thm. 3.7
      $I \leftarrow \square T([-R, R]) + [-\beta, \beta]$
      if $0 \in I$ **then**
        Add $(c_i, [c_{j+1}, c_{j-1}])$ to $\mathcal{S}$
      end if
    end for
  end for
  return $\mathcal{S}$
end procedure

Algorithm 2 Isolation function with subdivision

Input: A univariate polynomial P, a Chebyshev grid resolution integer N and two integers $0 \le i < j \le N-1$.

Output: Set of intervals $[c_{k+1}, c_k]$ containing all the roots of P in $[c_j, c_i]$, with $(c_k)_{k=0,\ldots,N-1}$ the Chebyshev nodes.

function ISOLATE_1D(P, N, i, j)
  if $\square P([c_j, c_i])$ contains 0 **then**
    if $i+1 < j$ **then** ▷ Split in two subintervals
      $k = \left\lfloor \frac{i+j}{2} \right\rfloor$
      $\mathcal{S}_1 \leftarrow$ ISOLATE_1D(P, N, i, k)
      $\mathcal{S}_2 \leftarrow$ ISOLATE_1D(P, N, $k+1$, j)
      return $\mathcal{S}_1 \cup \mathcal{S}_2$
    else
      return $\{[c_{i+1}, c_i]\}$
    end if
  else
    return $\emptyset$
  end if
end function

Algorithm 3 Multipoint partial evaluation with subdivision

Input: A bivariate polynomial $P(X, Y) = \sum_{i,j} a_{i,j} X^i Y^j$ with $\mathbf{a} \in \mathbb{R}^{(d+1)\times(d+1)}$ and a Chebyshev grid resolution integer $N > d > 0$.

Output: A set of vertical segments containing all the vertical crossed segments and contained in the set of candidate segments (Definition 4.2)

procedure SUBDIVISION(P, N)
  $\mathbf{d} \in \mathbb{R}^{N\times d}$
  for $k \leftarrow 0$ **to** d **do** ▷ Partial evaluations $\sum_{j=0}^{d} d_{i,j} Y^j = P(c_i, Y)$
    $d_{0,k},\ldots,d_{N-1,k} \leftarrow \square\text{FME}((a_{0,k},\ldots,a_{d,k}), N)$ ▷ N-point evaluation (Def. 3.5)
  end for
  $\mathcal{S} \leftarrow \emptyset$
  for $i \leftarrow 0$ **to** $N-1$ **do** ▷ Subdivision in vertical fibers $X = c_i$
    $S_Y \leftarrow$ ISOLATE_1D($\sum_k d_{i,k} Y^k$, N, 0, $N-1$)
    for $I_Y \in S_Y$, Add (c_i, I_Y) to $\mathcal{S}$
  end for
  return $\mathcal{S}$
end procedure

of degree d. Using a classical Horner evaluation, each evaluation is in $O(d)$. The complexity for all the fibers is thus $O(NdT)$. □

5 EXPERIMENTS

We present experiments for the 2-pass versions of our two algorithms for the drawing of algebraic curves (Section 5.1) and compare them to state-of-the-art software (Section 5.2).

Our algorithms are implemented in Python and use the interval arithmetic of Arb [19] for Python. We fix $m = 3$ in Algorithm 1. Among the state-of-the-art implementations for drawing implicit curves, we have selected ImplicitEquations [18], the marching squares from scikit and the implicit function plotting of MATLAB. ImplicitEquations is based on an algorithm of Tupper [40] and returns an enclosure of the algebraic curve. Contrary to our algorithm it misses no component. It distinguishes pixels where there is no solution, pixels where there is at least one solution and pixels where there may or may not be solutions (undecided by the algorithm). An implementation of the marching squares algorithm [26] is provided by `skimage.measure.find_contours` [42]. It takes as input the evaluations on the grid that we compute using `polyval1d` from the `numpy` package [15]. We also examine the behavior of `fimplicit` from the MATLAB, for which we were not able to find a documentation on the algorithm. All the measures are CPU times in seconds.

Our dataset is composed of weighted random polynomials of the form $\sum_{0\le i+j\le d} w_{i,j} a_{i,j} X^i Y^j$ of total degree d where the $a_{i,j}$ are

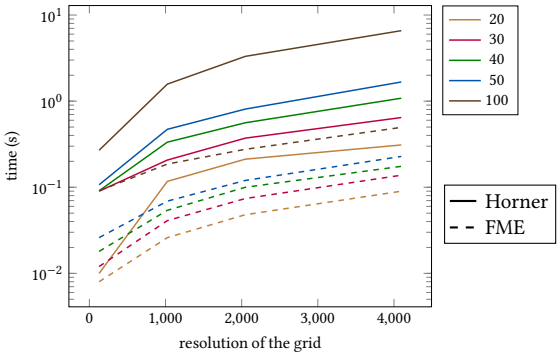
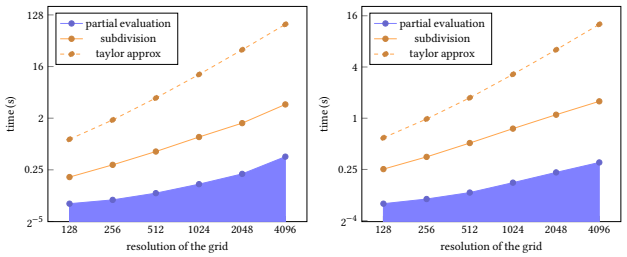


Figure 5: Computation times of the □FME and the interval Horner evaluation for the partial evaluation step. The color in the legend indicates the degree of Kac polynomials. Dotted lines correspond to the □FME and solid lines to the interval Horner scheme.



(a) *random_20_kac* **(b) *random_20_kss***

Figure 6: Cumulative time for a polynomial of total degree 20 weighted according to the two families.

uniformly distributed in $[-100, 100]$ and the weights are $w_{i,j} = 1$ for Kac polynomials, $w_{i,j} = \sqrt{\frac{d!}{i!j!(d-i-j)!}}$ for the Kostlan-Shub-Smale (KSS) polynomials [37]. Our input polynomials are named *random_d_weight* where d is the total degree and *weight* is either "kac" or "kss" depending on the family of the polynomial.

5.1 Comparison of our two approaches

Figure 5 experimentally verifies the relevance of the FME, it displays the computation times of the certified partial evaluation of our polynomials using FME, with a complexity of $O(d^3 + dN\log_2(N))$, and using Horner's method, with a complexity of $O(d^2N)$. For the range of values we are interested in, specifically $10 \leq d \leq 100$ and $500 < N$ the FME is always faster. This justifies the use of the FME despite the preliminary costly change of basis.

Figures 6 show log-log graphs for different input polynomials for Algorithm 1 with Taylor approximation and Algorithm 3 with subdivision that both share a common partial evaluation step. The slopes of these plots give the power in the complexities with respect to N. In these examples, the subdivision is faster than the Taylor approximation. Moreover, the slopes are around 2 for Algorithm 1 and around 1 for Algorithm 3. This confirms the expected result from the complexity analysis of Theorems 4.5 and 4.7, indeed with $d^2 < N\log_2(N)$ and $T = O(\log_2(N))$, the complexities are respectively $O(N^2\log_2(N))$ and $O(dN\log_2(N))$.

5.2 Comparison to state-of-the-art implementations

For our measures we choose to compute and save the results in PNG files. In Tables 6 and 7, all the implementations are tested on *random_20_kac* and *random_100_kac*. The methods ImplicitEquations, Taylor and Subdivision, which provide some guarantees are highlighted in orange. For ImplicitEquations and MATLAB, we stopped the computation after 900 seconds. As we already saw, Algorithm 1 is slower than Algorithm 3. Moreover, in these tables, the subdivision is slightly faster than scikit up to a resolution of 1024, despite the fact that our code may be slowed by Python limitations. Tables 8, 9 and 10, for *random_20_kss*, *random_40_kac* and *random_40_kss* lead to the same conclusions.

Figure 8 presents the plots of *random_20_kac* with Algorithm 3 and ImplicitEquations. Our algorithm returns a tighter enclosure. Figure 7 presents the plots of the KSS polynomial *random_20_kss* for all implementations, except ImplicitEquations, at a higher resolution of $N = 1024$. Compared to the plots of Figure 8, the non-uniformity of the Chebyshev grid is no longer visible at this resolution and the outputs look similar.

The marching squares from scikit and Algorithm 3 are the fastest for all the resolutions that we have tested. So, we test them for even higher resolutions in Tables 3 and 4. For Kac polynomials, the two methods are comparable up to a resolution of 8162 and our method becomes faster for higher resolutions. On the other hand, our method faces stability issues for KSS polynomials when the resolution and the degree increase: degree 20 and 30 polynomials are computed faster, but not the degree 40 one. This is explained by the sensitivity of the IDCT to the size of the input coefficients. The error bound presented in Table 2 becomes insufficient to control the drawing. Indeed, for *random_40_kss* with $N = 8192$, Theorem 3.4 yields $\|x - \widehat{x}\|_\infty \lesssim 1$, because the magnitude of the coefficients ranges from 1 to 10^{19} due to the KSS weights. For completeness, we also included in Table 5 the timings for Isotop,[1] a state-of-the-art software based on CAD [9], that computes a drawing with a guaranteed topology and a low-resolution. We have also tested the curve $\text{dfold}_{8,1}$ from Challenge 13 from Oliver Labs [24]. It contains an 8-fold singularity and has high tangencies between branches. For this curve, Isotop takes about one second while our subdivision method takes 10*s*, 39*s* and 193*s* for respective resolutions of 1024, 4096 and 16384, in particular, it fails to discard pixels around the singularity.

Our approach combining FME and subdivision proves to be competitive in our experiments. Even though our guarantee is a weaker than the one provided by ImplicitEquations, our algorithm is faster. For all polynomials but *random_40_kss*, it is also significantly faster than the marching squares from scikit for high resolutions ($N \geq 16384$), and has similar timings for the lower resolutions. The speed limitation of our implementation could be due to the Python language. We also think that the techniques that we presented could be used to speed-up existing algorithms such as marching squares or subdivision approaches.

[1] https://isotop.gamble.loria.fr

REFERENCES

[1] Eric Berberich, Pavel Emeliyanenko, Alexander Kobel, and Michael Sagraloff. Exact symbolic–numeric computation of planar algebraic curves. *Theoretical Computer Science*, 491:1–32, 2013. URL: https://www.sciencedirect.com/science/article/pii/S0304397513002983, doi:https://doi.org/10.1016/j.tcs.2013.04.014.

[2] Jean-Daniel Boissonnat and Monique Teillaud, editors. *Effective Computational Geometry for Curves and Surfaces*. Springer-Verlag, Mathematics and Visualization, 2006.

[3] Jean-Daniel Boissonnat and Mathijs Wintraecken. The topological correctness of pl approximations of isomanifolds. *Foundations of Computational Mathematics*, Jul 2021. doi:10.1007/s10208-021-09520-0.

[4] Alin Bostan, Bruno Salvy, and Éric Schost. Power series composition and change of basis. In *Proceedings of the Twenty-First International Symposium on Symbolic and Algebraic Computation*, ISSAC '08, page 269–276, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1390768.1390806.

[5] Nicolas Brisebarre, Mioara Joldes, Jean-Michel Muller, Ana-Maria Nanes, and Joris Picot. Error analysis of some operations involved in the cooley-tukey fast fourier transform. *ACM Trans. Math. Softw.*, 46(2):11:1–11:27, 2020. doi:10.1145/3368619.

[6] M Burr, F Krahmer, and Chee Yap. Continuous amortization: A non-probabilistic adaptive analysis technique. In *Electronic colloquium on computational complexity (ECCC), TR09 (136)*, 2009.

[7] Michael A. Burr. Continuous amortization and extensions: With applications to bisection-based root isolation. *Journal of Symbolic Computation*, 77:78–126, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0747717116000080, doi:https://doi.org/10.1016/j.jsc.2016.01.007.

[8] Michael A. Burr, Shuhong Gao, and Elias Tsigaridas. The complexity of an adaptive subdivision method for approximating real curves. In *Proceedings of the 2017 ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '17, pages 61–68, New York, NY, USA, 2017. ACM. URL: http://doi.acm.org/10.1145/3087604.3087654, doi:10.1145/3087604.3087654.

[9] Jinsan Cheng, Sylvain Lazard, Luis Peñaranda, Marc Pouget, Fabrice Rouillier, and Elias Tsigaridas. On the topology of real algebraic plane curves. *Mathematics in Computer Science*, 4(1):113–137, Nov 2010. doi:10.1007/s11786-010-0044-3.

[10] Felipe Cucker, Alperen A. Ergür, and Josue Tonelli-Cueto. Plantinga-vegter algorithm takes average polynomial time. In *Proceedings of the 2019 on International Symposium on Symbolic and Algebraic Computation*, ISSAC '19, pages 114–121, New York, NY, USA, 2019. ACM. URL: http://doi.acm.org/10.1145/3326229.3326252, doi:10.1145/3326229.3326252.

[11] Pavel Emeliyanenko, Eric Berberich, and Michael Sagraloff. Visualizing arcs of implicit algebraic curves, exactly and fast. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*, ISVC '09, page 608–619, Berlin, Heidelberg, 2009. Springer-Verlag. doi:10.1007/978-3-642-10331-5_57.

[12] Sarah F. F. Gibson. Constrained elastic surface nets: Generating smooth surfaces from binary segmented data. In William M. Wells, Alan Colchester, and Scott Delp, editors, *Medical Image Computing and Computer-Assisted Intervention — MICCAI'98*, pages 888–898, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[13] A. Gomes, I. Voiculescu, J. Jorge, B. Wyvill, and C. Galbraith. *Implicit Curves and Surfaces: Mathematics, Data Structures and Algorithms*. Springer London, 2009. URL: https://books.google.fr/books?id=mEmzjKMDlcAC.

[14] Stef Graillat and Valérie Ménissier-Morain. Compensated Horner scheme in complex floating point arithmetic. In *Proceedings, 8th Conference on Real Numbers and Computers*, pages 133–146, Santiago de Compostela, Spain, July 2008. URL: https://hal.archives-ouvertes.fr/hal-01300860.

[15] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi:10.1038/s41586-020-2649-2.

[16] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, second edition, 2002. URL: https://epubs.siam.org/doi/abs/10.1137/1.9780898718027, arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9780898718027, doi:10.1137/1.9780898718027.

[17] Kai Hormann, Lucas Kania, and Chee Yap. Novel range functions via taylor expansions and recursive lagrange interpolation with application to real root isolation. In *Proceedings of the 2021 on International Symposium on Symbolic and Algebraic Computation*, ISSAC '21, page 193–200, New York, NY, USA, 2021. Association for Computing Machinery. doi:10.1145/3452143.3465532.

[18] ImplicitEquations: Julia package to facilitate graphing of implicit equations and inequalities. https://github.com/jverzani/ImplicitEquations.jl.

[19] F. Johansson. Arb: a C library for ball arithmetic. *ACM Communications in Computer Algebra*, 47(4):166–169, 2013.

[20] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 339–346, New York, NY, USA, 2002. Association for Computing Machinery. doi:10.1145/566570.566586.

[21] Alexander Kobel, Fabrice Rouillier, and Michael Sagraloff. Computing real roots of real polynomials ... and now for real! In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, page 303–310, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2930889.2930937.

[22] Alexander Kobel and Michael Sagraloff. On the complexity of computing with planar algebraic curves. *J. Complex.*, 31(2):206–236, 2015. doi:10.1016/j.jco.2014.08.002.

[23] Alexander Kobel and Michael Sagraloff. Fast approximate polynomial multipoint evaluation and applications, 2016. arXiv:https://arxiv.org/abs/1304.8069v2 [cs.NA]. arXiv:1304.8069.

[24] O. Labs. A list of challenges for real algebraic plane curve visualization software. In *Nonlinear Computational Geometry*, volume IMA 151, pages 137–164. Springer, 2009.

[25] Long Lin and Chee Yap. Adaptive isotopic approximation of nonsingular curves: the parameterizability and nonlocal isotopy approach. *Discrete & Computational Geometry*, 45(4):760–795, Jun 2011. doi:10.1007/s00454-011-9345-9.

[26] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21:163–169, August 1987. URL: http://doi.acm.org/10.1145/37402.37422, doi:http://doi.acm.org/10.1145/37402.37422.

[27] J. Makhoul. A fast cosine transform in one and two dimensions. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(1):27–34, 1980. doi:10.1109/TASSP.1980.1163351.

[28] J.C. Mason and D.C. Handscomb. *Chebyshev Polynomials*. CRC Press, 2003.

[29] Ramon E Moore, R Baker Kearfott, and Michael J Cloud. *Introduction to interval analysis*. Siam, 2009.

[30] Arnold Neumaier. *Interval methods for systems of equations*. Cambridge University Press, 1990. URL: http://www.loc.gov/catdir/toc/cam041/89070812.html.

[31] V.Y. Pan. New fast algorithms for polynomial interpolation and evaluation on the chebyshev node set. *Computers & Mathematics with Applications*, 35(3):125–129, 1998. URL: https://www.sciencedirect.com/science/article/pii/S0898122197002836, doi:https://doi.org/10.1016/S0898-1221(97)00283-6.

[32] Simon Plantinga and Gert Vegter. Isotopic approximation of implicit curves and surfaces. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP '04, pages 245–254, New York, NY, USA, 2004. ACM. URL: http://doi.acm.org/10.1145/1057432.1057465, doi:10.1145/1057432.1057465.

[33] H. Ratschek and J. Rokne. *Computer Methods for the Range of Functions*. Computers and Their Applications. E. Horwood, 1984. URL: https://books.google.fr/books?id=v18ZAQAAIAAJ.

[34] Michael Sagraloff. A general approach to isolating roots of a bit-stream polynomial. *Mathematics in Computer Science*, 4(4):481, Oct 2011. doi:10.1007/s11786-011-0071-8.

[35] Michael Sagraloff and Kurt Mehlhorn. Computing real roots of real polynomials. *Journal of Symbolic Computation*, 73:46–86, 2016. URL: https://www.sciencedirect.com/science/article/pii/S0747717115000292, doi:https://doi.org/10.1016/j.jsc.2015.03.004.

[36] Arnold Schönhage. Asymptotically fast algorithms for the numerical multiplication and division of polynomials with complex coefficients. In Jacques Calmet, editor, *Computer Algebra*, pages 3–15, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[37] Michael Shub and Steve Smale. Complexity of bezout's theorem ii volumes and probabilities. In Frédéric Eyssette and André Galligo, editors, *Computational Algebraic Geometry*, pages 267–285, Boston, MA, 1993. Birkhäuser Boston.

[38] John M. Snyder. Interval analysis for computer graphics. *SIGGRAPH Comput. Graph.*, 26(2):121–130, July 1992. URL: http://doi.acm.org/10.1145/142920.134024, doi:10.1145/142920.134024.

[39] Volker Stahl. *Interval Methods for Bounding the Range of Polynomials and Solving Systems of Nonlinear Equations*. PhD thesis, Johannes Kepler University, Linz, Austria, 1995.

[40] Jeff Tupper. Reliable two-dimensional graphing methods for mathematical formulae with two free variables. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, page 77–86, New York, NY, USA, 2001. Association for Computing Machinery. doi:10.1145/383259.383267.

[41] Joris van der Hoeven. Fast composition of numeric power series. Technical Report 2008-09, Université Paris-Sud, Orsay, France, 2008.

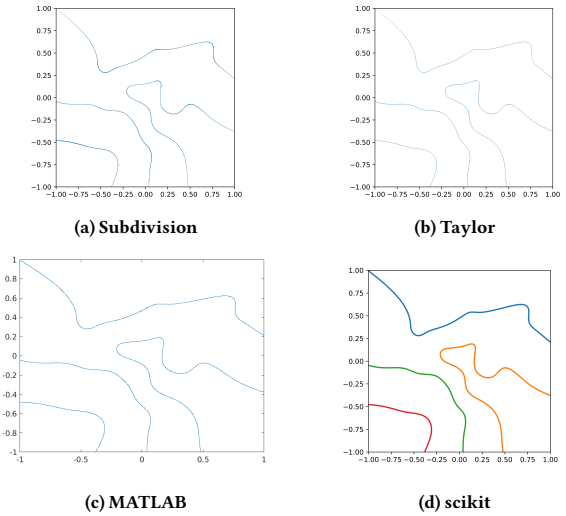
[42] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. doi:10.7717/peerj.453.

[43] Joachim von zur Gathen and Jürgen Gerhard. Fast algorithms for taylor shifts and certain difference equations. In *Proceedings of the 1997 International Symposium on Symbolic and Algebraic Computation*, ISSAC '97, page 40–47, New York, NY, USA, 1997. Association for Computing Machinery. doi:10.1145/258726.258745.

[44] Joachim von zur Gathen and Jürgen Gerhard. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013. doi:10.1017/CBO9781139856065.

Table 5: Computation times of Isotop for different families of polynomials.

random_20_kac	2.3	*random_100_kac*	>2000
random_30_kac	18	*random_20_kss*	12
random_40_kac	81	*random_30_kss*	183
random_50_kac	1603	*random_40_kss*	688



(a) Subdivision (b) Taylor

(c) MATLAB (d) scikit

Figure 7: *random_20_kss*, $N = 1024$

Table 3: Computation times of our subdivision algorithm for different families of polynomials with respect to the resolution.

N	128	256	512	1024	2048	4096	8192	16384	32768
random_20_kac	3.9	4.2	4.4	4.9	6.0	8.1	13	21	40
random_30_kac	4.1	4.1	4.3	4.7	5.5	7.3	11	18	33
random_40_kac	4.1	4.2	4.5	5.0	6.0	8.3	13	22	42
random_50_kac	5.3	4.2	4.4	5.1	6.2	8.5	13	23	44
random_100_kac	4.6	4.8	5.0	5.7	7.1	9.7	16	28	53
random_20_kss	4.1	4.3	4.4	5.2	6.1	9.2	15	26	48
random_30_kss	4.2	4.4	4.7	5.5	7.2	11	18	33	67
random_40_kss	4.2	4.5	5.1	6.2	8.7	15	36	145	718

Table 4: Computation times of scikit for different families of polynomials with respect to the resolution.

N	128	256	512	1024	2048	4096	8192	16384	32768
random_20_kac	5.9	5.8	5.8	5.9	6.2	7.2	11	29	83
random_30_kac	6.6	5.8	5.8	5.9	6.2	7.4	12	32	100
random_40_kac	6.5	5.9	5.8	5.8	6.3	7.6	12	35	116
random_50_kac	6.5	5.7	5.7	5.9	6.2	7.8	13	36	132
random_100_kac	6.4	5.7	5.8	6.0	6.5	8.5	17	49	232
random_20_kss	6.6	5.8	5.9	5.8	6.2	7.3	11	29	78
random_30_kss	6.7	5.9	5.8	5.8	6.3	7.4	12	32	102
random_40_kss	6.6	5.9	5.9	5.6	6.3	7.7	12	35	114

Table 6: Computation times for *random_20_kac* (in seconds), with our implementations highlighted in orange.

	N	128	256	512	1024	2048	4096
Method	scikit	5.8	5.7	5.8	5.9	6.3	7.3
	MATLAB	12	19	49	167	636	>900
	ImplictEquations	281	599	>900	>900	>900	>900
	taylor	4.5	5.8	11	26	87	332
	subdivision	3.7	4.2	4.4	4.9	5.9	8.1

Table 7: Computation times for *random_100_kac* (in seconds), with our implementations highlighted in orange.

	N	128	256	512	1024	2048	4096
Method	scikit	6.2	5.7	5.8	6.2	6.6	9.4
	MATLAB	79	283	>900	>900	>900	>900
	ImplicitEquations	>900	>900	>900	>900	>900	>900
	taylor	7.4	11	20	46	128	413
	subdivision	6.1	4.7	5.1	5.7	7.0	9.8

Table 8: Computation times for *random_20_kss* (in seconds), with our implementations highlighted in orange.

	N	128	256	512	1024	2048	4096
Method	scikit	5.8	5.8	5.8	5.9	6.2	7.1
	MATLAB	11	18	48	165	638	>900
	ImplictEquations	693	>900	>900	>900	>900	>900
	taylor	4.5	5.8	10	26	86	324
	subdivision	4.1	4.2	4.4	5.1	6.5	9.1

Table 9: Computation times for *random_40_kac* (in seconds), with our implementations highlighted in orange.

	N	128	256	512	1024	2048	4096
Method	scikit	5.8	6.0	6.0	6.1	6.3	7.5
	MATLAB	20.3	48.7	169	651	>900	>900
	ImplicitEquations	>900	>900	>900	>900	>900	>900
	taylor	5.0	6.6	12	30	94	345
	subdivision	4.0	4.2	4.5	5.0	6.1	8.4

Table 10: Computation times for *random_40_kss* (in seconds), with our implementations highlighted in orange.

	N	128	256	512	1024	2048	4096
Method	scikit	5.8	5.8	5.8	6.0	6.3	7.6
	MATLAB	19	50	171	656	>900	>900
	ImplicitEquations	>900	>900	>900	>900	>900	>900
	taylor	5.0	6.7	12	30	96	349
	subdivision	4.2	4.4	5.0	6.2	8.7	15

(a) Subdivision

(b) ImplicitEquations, where red pixels are undecided.

Figure 8: *random_20_kac*, $N = 128$