Numerical Algorithms
https://doi.org/10.1007/s11075-019-00726-6

ORIGINAL PAPER

Symbolic-numeric integration of rational functions

Robert H. C. Moir[1] · **Robert M. Corless**[1] · **Marc Moreno Maza**[1] · **Ning Xie**[2]

Received: 12 October 2018 / Accepted: 6 May 2019 / Published online: 29 May 2019

Abstract
We consider the problem of symbolic-numeric integration of symbolic functions, focusing on rational functions. Using a hybrid method allows the reliable yet efficient computation of symbolic antiderivatives while avoiding issues of ill-conditioning to which numerical methods are susceptible. We propose two alternative methods for exact input that compute the rational part of the integral using Hermite reduction and then compute the transcendental part two different ways using a combination of exact integration and efficient numerical computation of roots. The symbolic computation is done within BPAS, or Basic Polynomial Algebra Subprograms, which is a highly optimized environment for polynomial computation on parallel architectures, while the numerical computation is done using the highly optimized multiprecision rootfinding package MPSOLVE. We provide for both algorithms computable expressions for the first-order term of a structured forward and backward error and show how, away from singularities, tolerance proportionality is achieved by adjusting the precision of the rootfinding tasks.

Keywords Symbolic integration · Symbolic-numeric algorithms · Rational functions

1 Introduction

In this paper, we consider two algorithms for the approximate symbolic integration of univariate rational functions in $\mathbb{Q}(x)$ using a combination of symbolic and numerical methods. We provide in Section 4 a forward and backward error analysis of the symbolic result, providing computable expressions for the first-order term of a structured forward and backward error and showing that both the forward and backward errors are proportional to a user-supplied tolerance. Both algorithms have

✉ Robert H. C. Moir
robert@moir.net

[1] Ontario Research Centre for Computer Algebra, University of Western Ontario, London, ON N6A 3K7, Canada

[2] Huawei Technologies Corporation, Markham, ON L3R 5A4, Canada

been implemented in the open-source *Basic Polynomial Algebra Subprograms*, or BPAS, package (http://bpaslib.org), which is discussed in Section 5. The results of experiments on the implementations are discussed in Section 6. Although one of the algorithms emerges as stronger overall, both algorithms are seen to have advantages in different symbolic computing contexts.

1.1 Symbolic-numeric integration of rational functions

Hybrid symbolic-numeric integration of rational functions is interesting for several reasons. First, a formula, not a number or a computer program or subroutine, may be desired, perhaps for further analysis, such as by taking asymptotics. In this case one typically wants an exact symbolic answer, and for rational functions this is in principle always possible. However, an exact symbolic answer may be too cluttered with algebraic numbers or lengthy rational numbers to be intelligible or easily analyzed by further symbolic manipulation (see, e.g., Fig. 1). Discussing symbolic integration, Kahan [7] in his typically dry way gives an example "atypically modest, out of consideration for the typesetter", and elsewhere has rhetorically wondered: "Have you ever used a computer algebra system, and then said to yourself as screensful of answer went by, "I wish I hadn't asked."" Fateman has addressed rational integration [5], as have Noda and Miyahiro [8, 9], for this and other reasons.

Second, there is interest due to the potential to carry symbolic-numeric methods for rational functions forward to transcendental integration, since the rational function algorithm is at the core of more advanced algorithms for symbolic integration. Particularly in the context of *exact* input, which we assume, it can be desirable to have an intelligible approximate expression for an integral while retaining the exact expression of the integral for subsequent symbolic computation. The ability to do this is a feature of one of our algorithms that alternative approaches, particularly those based on partial fraction decomposition, do not share.

$$
\begin{aligned}
&-\frac{19}{84}\ln\left(x^2+\sqrt{2\sqrt{7}-5}\,x+\sqrt{7}\right)\sqrt{2\sqrt{7}-5}\,\sqrt{7}-\frac{7}{12}\ln\left(x^2+\sqrt{2\sqrt{7}-5}\,x+\sqrt{7}\right)\sqrt{2\sqrt{7}-5}\\
&+\frac{19}{42}\frac{\arctan\left(\dfrac{2x+\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)(2\sqrt{7}-5)\sqrt{7}}{\sqrt{2\sqrt{7}+5}}+\frac{7}{6}\frac{\arctan\left(\dfrac{2x+\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)(2\sqrt{7}-5)}{\sqrt{2\sqrt{7}+5}}\\
&-\frac{1}{7}\frac{\arctan\left(\dfrac{2x+\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)\sqrt{7}}{\sqrt{2\sqrt{7}+5}}+\frac{19}{84}\ln\left(x^2-\sqrt{2\sqrt{7}-5}\,x+\sqrt{7}\right)\sqrt{2\sqrt{7}-5}\,\sqrt{7}+\frac{7}{12}\ln\left(x^2\right.\\
&\left.-\sqrt{2\sqrt{7}-5}\,x+\sqrt{7}\right)\sqrt{2\sqrt{7}-5}+\frac{19}{42}\frac{\arctan\left(\dfrac{2x-\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)(2\sqrt{7}-5)\sqrt{7}}{\sqrt{2\sqrt{7}+5}}\\
&+\frac{7}{6}\frac{\arctan\left(\dfrac{2x-\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)(2\sqrt{7}-5)}{\sqrt{2\sqrt{7}+5}}-\frac{1}{7}\frac{\arctan\left(\dfrac{2x-\sqrt{2\sqrt{7}-5}}{\sqrt{2\sqrt{7}+5}}\right)\sqrt{7}}{\sqrt{2\sqrt{7}+5}}
\end{aligned}
$$

Fig. 1 MAPLE output for the integral $\int \frac{x^2-1}{x^4+5x^2+7}dx$

Besides intelligibility and retention of exact results, one might be concerned with numerical stability, or perhaps efficiency of evaluation. We consider stability issues in Sections 4 and 6. We remark that the algorithm we present here has quite superior numerical stability in many cases, and has good structured backward error and highly accurate answers, while providing the more intelligible answers we desire.

We emphasize that the goal of these algorithms is not to produce numerical values of definite integrals of rational functions, although it can be used for such. The goal is to produce an intelligible formula for the antiderivative which is correct in an approximate sense: the derivative of the answer produced will be another rational function near to the integrand, and, importantly, of the same form in that the denominator will have the correct degrees of its factors in its squarefree factorization and the residues in its partial fraction decomposition will also have the correct multiplicity.[1]

As indicated above, the combination of symbolic and numerical methods in the integration of rational functions is not new. Noda and Miyahiro [8, 9] developed a symbolic-numeric, or *hybrid*, method to integrate rational functions based on the use of the approximate symbolic algorithms for noisy data, numerical rootfinding and exact symbolic integration methods. Fateman [5] advocates a simpler hybrid approach, largely to produce a fast method that makes symbolic results more useful and more palatable, avoiding the "surd" or "RootOf" notation in the results of symbolic integrations. Both approaches work with the assumption that the input rational function has floating point coefficients.

For the existing symbolic-numeric algorithms for rational function integration, the approach is to be as sparing as possible in the use of symbolic algorithms to minimize their expense, in particular given that floating point input is assumed to be imprecise. In contrast, given that our working assumption is that the input rational function is *exact*, the present paper is dealing with a somewhat different problem, viz., the approach involves the injection of numerical methods into an otherwise purely symbolic context. As was mentioned above, the reasons such an approach is desirable include intelligibility, retention of exact results and stable or efficient evaluation. Since it is accuracy, speed and stability that matter in the context of scientific computing, a symbolic package that provides a suitable balance of these desiderata in a way that can be merged seamlessly with other scientific computations, as our implementation provides, has considerable advantages over CAS style symbolic computation with exact roots.

The usual approach to symbolic integration here begins with a rational function $f(x) = A(x)/B(x) \in \mathbb{Q}(x)$, with $\deg(A) < \deg(B)$ (ignoring any polynomial part, which can be integrated trivially) and computes an integral in two stages:

- rational part: computes a rational function C/D such that

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \int \frac{G(x)}{H(x)}dx, \tag{1}$$

[1]Note that strict preservation of the form of the integrand is not quite achieved for the PFD method described below, since the derivative cannot be simplified into this form without using approximate gcd. Thus, with exact computation, the degree of the numerator and denominator of the nearby integrand is larger in general than the exact integrand.

where the integral on the right hand side evaluates to a transcendental function (log and arctan terms);

- transcendental part: computes the second (integral) term of the expression (1) above yielding, after post-processing,

$$\int f(x)dx = \frac{C(x)}{D(x)} + \sum v_i \log(V_i(x)) + \sum w_j \arctan(W_j(x)), \qquad (2)$$

$V_i, W_j \in \mathbb{K}[x]$, with $\mathbb{K}$ being some algebraic extension of $\mathbb{Q}$.

In symbolic-numeric algorithms for this process, some steps are replaced by numeric or quasi-symbolic methods. Noda and Miyahiro use an approximate Horowitz method (involving approximate squarefree factorization) to compute the rational part and either the Rothstein-Trager (RT) algorithm or (linear/quadratic) partial fraction decomposition (PFD) for the transcendental part (see Section 2 for a brief review of these algorithms). The algorithm considered by Fateman avoids the two stage process and proceeds by numerical rootfinding of the denominator $B(x)$ (with multiplicities) and PFD to compute both rational and transcendental parts. In both cases, the working assumption is that the input uses floating point numbers that are subject to uncertainty or noise and the numerical algorithms use double precision.

Part of the power of symbolic algorithms is their ability to preserve structural features of a problem that may be very difficult to preserve numerically. Particularly given our focus on exact input, we are interested in preserving as much structure of the problem as possible if we are to resort to the use of numerical methods for rootfinding. Our implementation relies on the sophisticated rootfinding package MPSolve (http://numpi.dm.unipi.it/mpsolve), which provides a posteriori guaranteed bounds on the relative error of all the roots for a user-specified tolerance ε. To balance efficiency of computation and structure-preservation, we use more efficient symbolic algorithms where possible, such as the Hermite method for the rational part, and consider two methods of computing the transcendental part, one that computes the exact integral using the Lazard-Rioboo-Trager (LRT) method [2] followed by numerical approximation, and the other that uses a multiprecision PFD method to compute a nearby integrand that splits over $\mathbb{Q}$ and then performs a structured integration (for more details on the symbolic algorithms, see Section 2). The symbolic-numeric algorithms are discussed in Section 3.

The advantage of combining multiprecision numerical software with symbolic algorithms is that it allows the user to specify a tolerance on the error of the symbolic-numeric computation. This, together with *structured* backward and forward error analysis of the algorithm, then allows the result to be interpreted in a manner familiar to users of numerical software but with additional guarantees on structure-preservation. We provide such an analysis to give a posteriori computable expressions for the first-order term of the structured error of our algorithms in Section 4.

An interesting feature of the availability of a computable backward error for the algorithms is that it follows that the computed integral can be regarded as the *exact* integral of a slightly perturbed input integral, and, as stated previously, of the correct form (modulo the possible need for an approximate gcd). Insofar as the input rational function is the result of model construction or an application of approximation theory it is already subject to error, even though its coefficients are not floats. Thus, the input, though formally exact, is nevertheless still an approximation of a system or problem it represents. Assuming the model approximation error is small, this means that the rational function that best approximates the system or problem represented by the input is some nearby rational function in a small neighbourhood of $f(x)$, for example in the sense of the space determined by variation of the coefficients of f or in the sense determined by an integral norm, as we consider below. Where the backward error is shown to be small this therefore shows that the integral actually computed is also a nearby rational function within another small neighbourhood, whose size we have some control over by varying the input tolerance. In a manner similar to numerical analysis, then, by an appropriate choice of tolerance, we can ensure that the latter neighbourhood is smaller than the former, so that the numerical perturbation of the problem is smaller than the model approximation error. The upshot of this is that the use of backward error analysis shows how a symbolic-numeric algorithm can be compatible with the *spirit* of uncertain input, even if the input has non-float coefficients. That is, we are assuming that the modeling procedure got a rational function with the same structure as an exact model, even if its data is uncertain in other ways.

This shows how a backward error analysis can be useful even in the context of exact integration. In the general case of exact input, however, a backward error analysis alone is not enough. This is why we also provide a forward error analysis, to provide a posteriori assurance of a small forward error sufficiently far away from singularities in the integrand. We also provide such an analysis in Section 4.

Our algorithms may be adapted to take floating point input by using additional symbolic-numeric routines, such as approximate GCD and approximate squarefree factorization, in order to detect nearby problems with increased structure. Such an approach would shift the problem onto the *pejorative manifold*, i.e., the nearest most singular problem. This protects against ill-conditioning of the problem on account of the fact that ill-conditioning for roots of polynomials is the result of the ability of small perturbations to result in changes in multiplicities [6]. The symbolic-numeric structured PFD integration we propose already uses this approach for the transcendental part of the integral. The structured error analysis of our algorithms entails that the problem stays on remains on the pejorative manifold after the computation of the integral. Since there have been considerable advances in algorithms for approximate polynomial algebra since the time of writing of [9], such as the APATOOLS package of Zeng [12], the combination of error control and singular problem detection could yield a considerable advance over the early approach of Noda and Miyahiro.

2 Methods for exact integration of rational functions

We begin by reviewing symbolic methods for integrating rational functions.[2] Let $f \in \mathbb{R}(x)$ be a rational function over $\mathbb{R}$ with a denominator of positive degree. There exist polynomials $P, A, B \in \mathbb{R}[x]$ such that we have $f = P + A/B$ with $\gcd(A, B) = 1$ and $\deg(A) < \deg(B)$. Since P is integrated trivially, we ignore the general case and assume that $f = A/B$ with $\deg(A) < \deg(B)$. Furthermore, thanks to Hermite reduction, one can extract the rational part of the integral, leaving a rational function G/H, with $\deg(G) < \deg(H)$ and H squarefree, remaining to integrate. For the remainder of this section, then, we will assume that the function to integrate is given in the form G/H, with $\deg(G) < \deg(H)$ and H squarefree.

Partial-fraction decomposition (PFD) The partial fraction decomposition algorithm for rational functions in $\mathbb{R}(x)$ can be presented in different ways, depending on whether one admits complex numbers in expressions. We present a method based upon a complete factorization of the denominator over $\mathbb{C}$, followed by its conversion into an expression containing only constants from $\mathbb{R}$.

Consider the splitting of H expressed in the form

$$H = p \prod_{i=1}^{n} (x - \alpha_i) \prod_{j=n+1}^{n+m} \left[(x - (\alpha_j + i\,\beta_j))(x - (\alpha_j - i\,\beta_j)) \right],$$

separating real roots from complex conjugate pairs, where $p, \alpha_k, \beta_k \in \mathbb{R}$. Then, there exist a_k and b_k such that

$$\frac{G}{H} = \sum_{i=1}^{n} \frac{a_i}{x - \alpha_i} + \sum_{j=n+1}^{n+m} \left[\frac{a_j + i\,b_j}{(x - (\alpha_j + i\,\beta_j))} + \frac{a_j - i\,b_j}{(x - (\alpha_j - i\,\beta_j))} \right]. \tag{3}$$

The numerator quantities $c_k = a_k + i\,b_k$ corresponding to the roots $\gamma_k = \alpha_k + i\,\beta_k$ we call *residues* by analogy to complex analysis. Note that in the case here where H is squarefree, the residues can be computed by the formula $c_k = c(\gamma_k) = G(\gamma_k)/H'(\gamma_k)$.

The real root terms are easily integrated to yield terms of the form $a_i \log(x - \alpha_i)$. Extracting terms of the form $a_j \left[(x - (\alpha_j + i\,\beta_j))^{-1} + (x - (\alpha_j - i\,\beta_j))^{-1} \right]$ from (3) we obtain pairs of complex log terms that can be combined to form a single real log term of the form $a_j \log(x^2 - 2\alpha_j x + \alpha_j^2 + \beta_j^2)$. Extracting terms of the form $i\,b_j \left[(x - (\alpha_j + i\,\beta_j))^{-1} - (x - (\alpha_j - i\,\beta_j))^{-1} \right]$ from (3) and making use of the observation of Rioboo that

$$\frac{d}{dx} i \log\left(\frac{X + iY}{X - iY} \right) = \frac{d}{dx} 2 \arctan(X/Y), \tag{4}$$

[2]The following review is based in part on the ISSAC 1998 tutorial [3] and the landmark text book [2] of M. Bronstein.

for $X, Y \in \mathbb{R}[x]$ (see [2], pp. 59*ff.*), we obtain a term of the form $2b_j \arctan\left(\frac{\alpha_j - x}{\beta_j}\right)$.

Where there are repeated residues in the PFD, it is possible to combine terms of the integral together. The combination of logarithms with common a_k simply requires computing the product of their arguments. For the arctangent terms, the combination of terms with common b_k can be accomplished by iterated application of the rule

$$\arctan\left(\frac{X}{Y}\right) + \arctan\left(\frac{\alpha - x}{\beta}\right) \rightarrow \arctan\left(\frac{X(\alpha - x) - \beta Y}{Y(\alpha - x) + \beta X}\right), \tag{5}$$

which is based on the fact that $\log(X + i\,Y) + \log((\alpha - x) + i\,\beta) = \log((X(\alpha - x) - \beta Y) + i\,(Y(\alpha - x) + \beta X))$ and (4).

A major computational bottleneck of the symbolic algorithms based on a PFD is the necessity of factoring polynomials into irreducibles over $\mathbb{R}$ or $\mathbb{C}$ (and not just over $\mathbb{Q}$) thereby introducing algebraic numbers even if the integrand and its integral are both in $\mathbb{Q}(x)$. Unfortunately, introducing algebraic numbers may be necessary: any field containing an integral of $1/(x^2 + 2)$ contains $\sqrt{2}$ as well. A result of modern research are so-called *rational* algorithms that compute as much of the integral as can be kept within $\mathbb{Q}(x)$, and compute the minimal algebraic extension of $\mathbb{K}$ necessary to express the integral.

The Rothstein-Trager algorithm It follows from the PFD of G/H, i.e., $G/H = \sum_{i=1}^{n} c_i/(x - \gamma_i)$, $c_i, \gamma_i \in \mathbb{C}$, that

$$\int \frac{G}{H}\,dx = \sum_{i=1}^{\deg(H)} c_i \log(x - \gamma_i) \tag{6}$$

where the γ_i are the zeros of H in $\mathbb{C}$ and the c_i are the residues of G/H at the γ_i. Computing those residues without splitting H into irreducible factors is achieved by the Rothstein-Trager theorem, as follows. Since we seek roots of H and their corresponding residues given by evaluating $c = G/H'$ at the roots, it follows that the c_i are exactly the zeros of the *Rothstein-Trager resultant* $R := \text{resultant}_x(H, G - cH')$, where c here is an indeterminate. Moreover, the splitting field of R over $\mathbb{Q}$ is the minimal algebraic extension of $\mathbb{Q}$ necessary to express $\int f$ in the form given by Liouville's theorem, i.e., as a sum of logarithms, and we have

$$\int \frac{G}{H}\,dx = \sum_{i=1}^{m} \sum_{c|U_i(c)=0} c \log(\gcd(H, G - cH')) \tag{7}$$

where $R = \prod_{i=1}^{i=m} U_i^{e_i}$ is the irreducible factorization of R over $\mathbb{Q}$.

The Lazard-Rioboo-Trager algorithm Consider the subresultant pseudo-remainder sequence R_i, where $R_0 = R$ is the resultant (see p. 115 in [4]) of H and $G - cH'$ w.r.t. x. Observe that the resultant R is a polynomial in c of degree $\deg(H)$, the roots of which are the residues of G/H. Let $U_1 U_2^2 \cdots U_m^m$ be a square-free factorization of R. Then, we have

$$\int \frac{G}{H}\,dx = \sum_{i=1}^{m} \sum_{c|U_i(c)=0} c \log(\gcd(H, G - cH')), \tag{8}$$

which groups together terms of the PFD with common residue, as determined by the multiplicity of U_i in the squarefree factorization. We compute the inner sum $\sum_{c|U_i(c)=0} c \log(\gcd(H, G - cH'))$ as follows. If all residues of H are equal, there is a single nontrivial squarefree factor with $i = \deg(H)$ yielding $\sum_{c|U_i(c)=0} c \log(H)$; otherwise, that is, if $i < \deg(H)$, the sum is $\sum_{c|U_i(c)=0} c \log(S_i)$, where $S_i = \mathrm{pp}_x(R_k)$, where $\deg_x(R_k) = i$ and pp_x stands for primitive part w.r.t. x. Consequently, this approach requires isolating only the complex roots of the square-free factors $U_1, U_2, \ldots, U_m$, whereas methods based on the PFD require isolating the real or complex roots of the polynomial H, where $\deg(H) \geq \sum_i \deg(U_i)$. However, the coefficients of R (and possibly those of $U_1, U_2, \ldots, U_m$) are likely to be larger than those of H. Overall, depending on the example, the computational cost of root isolation may put at advantage any of those approaches in comparison with the others.

3 The algorithms

We consider two symbolic-numeric algorithms, both based on Hermite reduction for the rational part and using two distinct methods for the transcendental part, one based on structured partial fraction decomposition and the other the Lazard-Rioboo-Trager algorithm, both reviewed in Section 2. Following the notation used in (1), we assume the rational part C/D has been computed and we consider how the transcendental part is computed by the two methods. Both algorithms use MPSOLVE to control the precision on the root isolation step.

Following the notations used in (8), the LRT-based method proceeds by computing the subresultant chain $(R_0, R_1, \ldots)$ and deciding how to evaluate each sum $\sum_{c|U_i(c)=0} c \log(R_k)$, $\deg(R_k) = i$, by applying the strategy of Lazard, Rioboo and Trager. However, we compute the complex roots of the polynomials $U_1, U_2, \ldots, U_m$ numerically instead of representing them symbolically as in [10, 11]. Then, we evaluate each sum $\sum_{c|U_i(c)=0} c \log(R_k)$ by an algorithm adapted to this numerical representation of the roots. This method is presented as Algorithm 1 (see page 10).

The PFD-based method begins by computing numerically the roots γ_i of the denominator $H(x)$ and then computes exactly the resulting residues $c_i = c(\gamma_i) = G(\gamma_i)/H'(\gamma_i)$. The numerical rootfinding can destroy the structure of repeated residues, which we restore by detecting residues that differ by less than ε, the user-supplied tolerance. The resulting partial fraction decomposition can then be integrated using the structure-preserving strategy presented in Section 2 above. This strategy allows the algorithm to replicate the structure of the final output from the LRT algorithm as a sum of real logarithms and arctangents. This method is presented as Algorithm 2 (see page 10).

We remark that there can be an issue here in principle as a result of roots of H that are closer than ε. Given the properties of MPSOLVE, however, this is not an issue in practice, given the ability to compute residues exactly or with sufficiently high precision, because MPSOLVE isolates roots within regions where Newton's method converges quadratically. In the unlikely event of residues that are distinct but within ε of each other, the algorithm still results in a small error and is advantageous in terms

of numerical stability. This is because identifying nearby roots shifts the problem onto the pejorative manifold, as mentioned above.

Both methods take as input a univariate rational function $f(x) = A(x)/B(x)$ over $\mathbb{Q}$ with $\deg(B) > \deg(A)$, and a tolerance $\varepsilon > 0$. Both $A(x)$ and $B(x)$ are expressed in the monomial basis. They yield as output an expression

$$\int \hat{f}(x)\,dx = \frac{C}{D} + \sum v_i \log(V_i) + \sum w_j \arctan(W_j), \tag{9}$$

where $V_i, W_j \in \mathbb{Q}[x]$ and $\hat{f}(x)$ is the nearby integrand corresponding to the computed integral, along with a linear estimate of the forward and backward errors. The backward error on an interval $[a, b]$ is measured in terms of $\|\Delta f\|_\infty = \max_{a \le x \le b} |\Delta f(x)|$, where $\Delta f(x) = f(x) - \frac{d}{dx} \int \hat{f}(x)\,dx = f(x) - \hat{f}(x)$. The forward error on $[a, b]$ is measured in terms of $\| \int (f(x) - \hat{f}(x))\,dx \|_\infty = \| \int \Delta f(x) dx \|_\infty$, where $\int f(x)dx$ and $\int \hat{f}(x)dx$ are assumed to have the same constant of integration. Where f has no real singularities, the results in the following section provide error bounds over $\mathbb{R}$, and where f has real singularities the bounds can be used to determine how close to the singularity the error exceeds the tolerance.

The main steps of Algorithm 1 and Algorithm 2 are listed below, where the numbers between parentheses refer to lines of the pseudo-code below. Both algorithms begin with:

(1–4:) Decompose $\int f\,dx$ into $\frac{C}{D}$ (rational part) and $\int \frac{G}{H}\,dx$ (transcendental part) using Hermite reduction;

Algorithm 1 then proceeds with:

(5–6:) Compute symbolically the transcendental part $\int \frac{G}{H}\,dx = \sum_i \sum_{c|U_i(c)=0} c \cdot \log(S_i(t, x))$ using the Lazard-Rioboo-Trager algorithm; in the pseudo-code $\boldsymbol{U}$ is a vector holding the square-free factors of the resultant while $\boldsymbol{S}$ holds the primitive part of elements of the subresultant pseudo-remainder sequence corresponding to elements of $\boldsymbol{U}$, viz., such that corresponding to U_i is $S_i = \mathrm{pp}_x(R_k)$, where $\deg_x(R_k) = i$;

(7:) Compute the roots c_k of $U_i(c)$ numerically using MPSOLVE to precision ε.

(8–9:) Compute the log and arctan terms using symbolic post-processing in BPAS.

After Hermite reduction, Algorithm 2 continues with:

(5–6:) Compute the roots γ_k of $H(x)$ numerically using MPSOLVE to precision ε.

(7:) Compute the residues c_k of $G(x)/H(x)$ corresponding to the approximate roots of $H(x)$ and detect their identity within ε.

(8:) Compute identical residues within ε and then compute a correspondence φ (one-many relation) between a representative residue and its corresponding roots. φ correlates indices of selected elements of $\boldsymbol{c}$, the vector of residues, and indices of elements of $\boldsymbol{\gamma}$, the vector of roots.

(9–10:) Compute symbolically the transcendental part $\int \frac{\hat{G}}{\hat{H}}\,dx = \sum v_i \log(V_i) + \sum w_j \arctan(W_j)$ from the PFD of $\hat{G}(x)/\hat{H}(x)$.

Both algorithms complete the integration by processing the arctangent terms, which can be written as $\arctan\left(\frac{X}{Y}\right)$ or $\arctan(X, Y)$, for polynomials X and Y, to remove spurious singularities. This is accomplished with Rioboo's singularity removal (RSR) method (described in [2]), which is based on (4) and the extended Euclidean algorithm. The result is the conversion of the arctangent of a rational function or two-argument arctangent into a sum of arctangents of polynomials.

Algorithm 1 symbolicNumericIntegrateLRT(f,ε) $f \in \mathbb{Q}(x)$, $\varepsilon > 0$.

1: $(g, h) \leftarrow$ hermiteReduce(num(f), den(f)) // Note: $g, h \in \mathbb{Q}(x)$
2: $(Quo, Rem) \leftarrow$ euclideanDivide(num(h), den(h)) // Note: Quo, $Rem \in \mathbb{Q}[x]$
3: **if** $Quo \neq 0$ **then**
4: $\quad P \leftarrow$ integrate(Quo)
5: **if** $Rem \neq 0$ **then**
6: $\quad (\boldsymbol{U}, \boldsymbol{S}) \leftarrow$ integrateLogPart(Rem, den(h)) // Note: $\boldsymbol{U} = (U_i, 1 \leq i \leq m)$ and $\boldsymbol{S} = (S_i)$ are vectors with coefficients in $\mathbb{Q}[t]$ and $\mathbb{Q}[t, x]$ respectively
7: $\boldsymbol{c} \leftarrow$ rootsMP($\boldsymbol{U}$, ε) // Note: $\boldsymbol{c} = (c_k)$ are the roots of U_i, as returned by MPSOLVE
8: $(\boldsymbol{L}, \boldsymbol{A2}) \leftarrow$ logToReal($\boldsymbol{c}$, $\boldsymbol{S}$) // Note: $\boldsymbol{L}$ and $\boldsymbol{A2}$ are, respectively, vectors of logs and two-argument arctangent terms
9: $\boldsymbol{A} \leftarrow$ atan2ToAtan($\boldsymbol{A2}$)
10: **return** $(P, g, \boldsymbol{L}, \boldsymbol{A})$

Algorithm 2 symbolicNumericIntegratePFD(f,ε) $f \in \mathbb{Q}(x)$, $\varepsilon > 0$.

1: $(g, h) \leftarrow$ hermiteReduce(num(f), den(f)) // Note: $g, h \in \mathbb{Q}(x)$
2: $(Quo, Rem) \leftarrow$ euclideanDivide(num(h), den(h)) // Note: Quo, $Rem \in \mathbb{Q}[x]$
3: **if** $Quo \neq 0$ **then**
4: $\quad P \leftarrow$ integrate(Quo)
5: **if** $Rem \neq 0$ **then**
6: $\quad \boldsymbol{\gamma} \leftarrow$ rootsMP(den(h), ε) // Note: $\boldsymbol{\gamma} = (\gamma_k)$ are the roots of den(h), as returned by MPSOLVE
7: $\quad \boldsymbol{c} \leftarrow$ residues(Rem, den(h), $\boldsymbol{\gamma}$) // Note: $\boldsymbol{c} = (c_k)$ are the residues corresponding to the γ_i
8: $\quad \varphi \leftarrow$ residueRootCorrespondence($\boldsymbol{c}$, $\boldsymbol{\gamma}$, ε) // Note: $\varphi \subseteq \mathbb{N} \times \mathbb{N}$
9: $\quad (\boldsymbol{L}, \boldsymbol{A2}) \leftarrow$ integrateStructuredPFD($\boldsymbol{c}$, $\boldsymbol{\gamma}$, φ) // Note: $\boldsymbol{L}$ and $\boldsymbol{A2}$ are, respectively, vectors of logs and two-argument arctangent terms
10: $\quad \boldsymbol{A} \leftarrow$ atan2ToAtan($\boldsymbol{A2}$)
11: **return** $(P, g, \boldsymbol{L}, \boldsymbol{A})$

4 Analysis of the algorithms

We now consider the error analysis of the symbolic-numeric integration using LRT and PFD. We present a linear forward and backward error analysis for both methods.[3]

[3]Note that throughout this section we assume that the error for the numerical rootfinding for a polynomial $P(x)$ satisfies the relation $|\Delta r| \leq \varepsilon |r|$, where r is the value of the computed root and Δr is the distance in the complex plane to the exact root. This can be accomplished using MPSOLVE by specifying an error tolerance of ε. Given the way that MPSOLVE isolates and then approximates roots, the bound is generally satisfied by several orders of magnitude.

Theorem 1 (Computable Backward Error) *Given a rational function $f = A/B$ satisfying $deg(A) < deg(B)$, $gcd(A, B) = 1$ and input tolerance ε, Algorithm 1 and Algorithm 2 yield an integral of a rational function $\hat{f}$ such that for $\Delta f = f - \hat{f}$,*

$$\|\Delta f\|_\infty = \max_x \left| \sum_k Re\left(\Xi(x, r_k)\right) \right| + O(\varepsilon^2),$$

where the principal term is $O(\varepsilon)$, r_k ranges over the evaluated roots and the function Ξ defined below is computable. This expression for the backward error is finite on any closed, bounded interval not containing a root of $B(x)$.

We remark that the result is expressed in terms of an unspecified function $\Xi(x, r_k)$ both because the PFD-based and LRT-based methods result in different expressions and because the two methods compute roots of different quantities (roots of the denominator of the integrand for the PFD-based method, and roots of the Rothstein-Trager resultant for the LRT-based method).

Proof (PFD-based backward error) The PFD method begins by using Hermite reduction to obtain

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \int \frac{G(x)}{H(x)}\,dx, \tag{10}$$

where $H(x)$ is squarefree. Given the roots γ_i of $H(x)$, we may obtain the PFD of $G(x)/H(x)$, yielding

$$\frac{G(x)}{H(x)} = \sum_{i=1}^{\deg(H)} \frac{c_i}{x - \gamma_i}, \tag{11}$$

where $c_i = c(\gamma_i)$ with $c(x) = G(x)/H'(x)$. Taking account of identical residues, the expression (11) can then be integrated using the structured PFD algorithm described in Section 2. Since we approximate the roots of H, we replace the exact roots γ_i with the approximations $\hat{\gamma}_i$. This destroys the symmetry of the exactly repeated residues; thus, the (exact) c_i are modified in two ways: by evaluating $c(x)$ at $\hat{\gamma}_i$; and by adjusting the list of computed residues to restore symmetry, so that residues within ε of each other are coalesced. This strategy requires some method of selecting a single representative for the list of nearby residues; the error analysis then estimates the error on the basis of the error of this representative.[4] We then represent this adjusted computed list of residues by $\hat{c}_i$. Since the Hermite reduction and PFD are equivalent to a rewriting of the input function $f(x)$ as

$$f(x) = \frac{C'(x)}{D(x)} - \frac{C(x)D'(x)}{D(x)^2} + \sum_{i=1}^{\deg(H)} \frac{c_i}{x - \gamma_i},$$

[4]Note that we assume that ε is sufficiently small to avoid spurious identification of residues in this analysis. Even with spurious identification, however, the backward error analysis would only change slightly, viz., to use the maximum error among the nearby residues, rather than the error of the selected representative residue. A different but related issue occurs when several distinct residues are clustered within an interval of width of some small multiple of ε. This presents a more challenging situation that we reserve for future work, but discuss briefly in Section 7 below.

the modified input $\hat{f}(x)$ that Algorithm 2 integrates exactly is obtained from the above expression by replacing c_i and γ_i, respectively, with $\hat{c}_i$ and $\hat{\gamma}_i$.

To compute the backward error we first must compute the sensitivity of the residues to changes in the roots. Letting $\Delta\gamma_i = \gamma_i - \hat{\gamma}_i$, then to first order we find that

$$c_i = c(\gamma_i) = c(\hat{\gamma}_i) + c'(\hat{\gamma}_i)\Delta\gamma_i + O(\Delta\gamma_i^2),$$

where $c' = \frac{G'}{H'} - \frac{GH''}{H'^2}$. So the backward error for a given term of the PFD is

$$\begin{aligned}
\frac{c_i}{x-\gamma_i} - \frac{\hat{c}_i}{x-\hat{\gamma}_i} &= \frac{(c_i-\hat{c}_i)(x-\hat{\gamma}_i) + \hat{c}_i\Delta\gamma_i}{(x-\gamma_i)(x-\hat{\gamma}_i)} + O(\Delta\gamma_i^2) && (12)\\
&= \frac{c'(\hat{\gamma}_i)\Delta\gamma_i}{(x-\hat{\gamma}_i-\Delta\gamma_i)} + \frac{\hat{c}_i\Delta\gamma_i}{(x-\hat{\gamma}_i)(x-\hat{\gamma}_i-\Delta\gamma_i)} + O(\Delta\gamma_i^2) && (13)\\
&= \frac{c'(\hat{\gamma}_i)\Delta\gamma_i}{(x-\hat{\gamma}_i)} + \frac{\hat{c}_i\Delta\gamma_i}{(x-\hat{\gamma}_i)(x-\hat{\gamma}_i)} + O(\Delta\gamma_i^2). && (14)
\end{aligned}$$

Since any identified residues all approximate the same exact residue c_k, we use the error $c'(\gamma_k)$ for the residue $\hat{c}_k$ selected to represent the identical residues.

Now, because the rational part of the integral is computed exactly, only the PFD contributes to the backward error. Given that γ_i is an exact root of $H(x)$

$$H(\gamma_i) = 0 = H(\hat{\gamma}_i) + H'(\hat{\gamma}_i)\Delta\gamma_i + O(\Delta\gamma_i^2),$$

where $H(\hat{\gamma}_i) \neq 0$ unless the exact root is computed, and $H'(\gamma_i) \neq 0$ (and hence $H'(\hat{\gamma}_i) \neq 0$) because H is squarefree. Thus, we have that $\Delta\gamma_i = -H(\hat{\gamma}_i)/H'(\hat{\gamma}_i)$ to first order, where $|\Delta\gamma_i| \leq \varepsilon|\hat{\gamma}_i|$. We therefore find that

$$\Delta f = f - \hat{f} = -\sum_{i=1}^{\deg(H)} \left(\frac{c'(\hat{\gamma}_i)}{x-\hat{\gamma}_i} + \frac{\hat{c}_i}{(x-\hat{\gamma}_i)^2}\right)\frac{H(\hat{\gamma}_i)}{H'(\hat{\gamma}_i)} + O(\varepsilon^2). \tag{15}$$

Since the summand is a rational function depending only on x and $\hat{\gamma}_i$, for fixed x the imaginary parts resulting from complex conjugate roots will cancel, so that only the real parts of the summand contribute to the backward error. We therefore find a first-order expression of the backward error in the form of the theorem statement with

$$\boxed{\Xi(x, r_k) = \left(\frac{c'(r_k)}{x-r_k} + \frac{c(r_k)}{(x-r_k)^2}\right)\frac{H(r_k)}{H'(r_k)}},$$

where r_k ranges over the computed roots of $H(x)$. This expression is $O(\varepsilon)$ because $\frac{H(r_k)}{H'(r_k)}$ is $O(\varepsilon)$. □

Note that, to properly account for the adjusted residue, applying the formula for Ξ in the PFD case requires taking r_k to be the γ_k used to evaluate the representative residue.

Proof (LRT-based backward error) The LRT algorithm produces an exact integral of the input rational function in the form

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \sum_{i=1}^{n} \sum_{c \mid U_i(t)=0} c \cdot \log(S_i(c,x)). \tag{16}$$

Given a list $c_{ij} \in \mathbb{C}$, $1 \le j \le \deg(U_i)$ of roots of $U_i(t)$, we can express the integral in the form

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \sum_{i=1}^{n} \sum_{j=1}^{\deg(U_i)} c_{ij} \cdot \log(S_i(c_{ij},x)),$$

where n is the number of nontrivial squarefree factors of $\text{resultant}_x(H, G - cH')$. Taking the derivative of this expression we obtain an equivalent expression of the input rational function as

$$f(x) = \frac{C'(x)}{D(x)} - \frac{C(x)D'(x)}{D(x)^2} + \sum_{i=1}^{n} \sum_{j=1}^{\deg(U_i)} c_{ij} \frac{\frac{\partial S_i(c_{ij},x)}{\partial x}}{S_i(c_{ij},x)}. \tag{17}$$

The modified input $\hat{f}(x)$ that Algorithm 1 integrates exactly is obtained from this expression by replacing the exact roots c_{ij} with their approximate counterparts $\hat{c}_{ij}$.

To compute the backward error, we must compute the sensitivity of (17) to changes of the roots. Considering f as a function of the parameters c_{ij}, and letting $\Delta c_{ij} = c_{ij} - \hat{c}_{ij}$, the difference between the exact root and the computed root, we find by taking partial derivatives with respect to the c_{ij} that

$$\begin{aligned} &f(x, c_{11}, \ldots, c_{n\deg(U_n)}) = f(x, \hat{c}_{11}, \ldots, \hat{c}_{n\deg(U_n)}) \\ &+ \sum_{i=1}^{n} \sum_{j=1}^{\deg(U_i)} \left[\frac{\frac{\partial S_i(c,x)}{\partial x}}{S_i(c,x)} + c \left(\frac{\frac{\partial^2 S_i(c,x)}{\partial x \partial c}}{S_i(c,x)} - \frac{\frac{\partial S_i(c,x)}{\partial x}\frac{\partial S_i(c,x)}{\partial c}}{S_i(c,x)^2} \right) \right] \Bigg|_{c=\hat{c}_{ij}} \Delta c_{ij} + O(\Delta c_{ij}^2). \end{aligned} \tag{18}$$

Since $f(x, \hat{c}_{11}, \ldots, \hat{c}_{n\deg(U_n)}) = \hat{f}(x)$, letting the rational function in square brackets be denoted by $\xi_i(c,x)$, we have that

$$\Delta f = f - \hat{f} = \sum_{i=1}^{n} \sum_{j=1}^{\deg(U_i)} \xi_i(\hat{c}_{ij}, x) \Delta c_{ij} + O(\Delta \hat{c}_{ij}^2).$$

Given that $U_i(c_{ij}) = 0 = U_i(\hat{c}_{ij}) + U_i'(\hat{c}_{ij})\Delta c_{ij} + O(\Delta c_{ij}^2)$, we have that $\Delta c_{ij} = -U_i(\hat{c}_{ij})/U_i'(\hat{c}_{ij})$ to first order, where $|\Delta c_{ij}| \le \varepsilon |\hat{c}_{ij}|$. Since, as for the PFD case, the

imaginary terms from complex roots cancel, we therefore find a first-order expression for the backward error in the form required by the theorem with

$$\boxed{\Xi(x, r_k) = \left[\frac{\frac{\partial S_i(r,x)}{\partial x}}{S_i(r,x)} + r\left(\frac{\frac{\partial^2 S_i(r,x)}{\partial x \partial r}}{S_i(r,x)} - \frac{\frac{\partial S_i(r,x)}{\partial x}\frac{\partial S_i(r,x)}{\partial r}}{S_i(r,x)^2}\right)\right]\Bigg|_{r=r_k} \frac{U_i(r_k)}{U_i'(r_k)}},$$

where r_k runs over the roots $\hat{c}_{ij}$. This expression is $O(\varepsilon)$ because $\frac{U_i(r_k)}{U_i'(r_k)}$ is $O(\varepsilon)$. □

Note that the backward error is structured, because the manner in which the integral is computed preserves certain structure in the integrand for both the LRT-based Algorithm 1 and the PFD-based Algorithm 2. For the LRT-based method, the fact that the residues are computed without perturbing the roots entails that the use of Hermite reduction guarantees that the roots of the denominator of $\hat{f}(x)$ have the same multiplicity as the roots of the denominator of f. The use of subresultants and the Rothstein-Trager resultant in Algorithm 1 also ensures that the approximated residues have the same multiplicity as the exact residues. For the PFD-based method, the approximation of the roots of $H(x)$ entails that the roots in the denominators of the derivatives of the rational and transcendental parts are slightly different. Thus, the denominator root multiplicity is only preserved approximately, which is why approximate gcd is needed to simplify the derivative of the integral. The identification of nearby computed residues in Algorithm 2, however, ensures that the multiplicity of residues in the PFD of G/H is preserved exactly, so that for the transcendental part the PFD of f and $\hat{f}$ have the same structure. The preservation of residue multiplicity translates into higher degree arguments in the log and arctan terms of the integral than would be obtained by a standard PFD algorithm, leading to structured forward error as well.

It is important to reflect on the behaviour of these error terms $\Xi(x, r_k)$ near singularities of the integrand, which correspond to real roots of $H(x)$ (and $B(x)$). For both algorithms, Ξ contains a particular polynomial in the denominator that evaluates to zero at the real roots, specifically $x - \gamma_i$ and $S_i(c_{ij}, x)$ for the PFD-based and LRT-based methods, respectively. In both cases, the expression of Ξ has a term with that particular polynomial squared, which therefore asymptotically dominates the value of the error term near the singularity. This fact is important for efficient computation of the size of the error term near a singularity, since the scaling behaviour can be used to quickly locate the boundary around the singularity where the error starts to exceed the tolerance. Our implementation discussed in Section 5 uses this scaling to compute such boundaries.

We turn now to the consideration of forward error of the algorithms. We note that a full forward error analysis on this problem has subtleties on account of the numerical sensitivities of the log function. This does not affect the validity of the forward error results to follow (that contain both a log term and a simple pole) because near singularities the log term is dwarfed by the pole term, so can be safely ignored in the computation of singularity boundaries. It is a concern, however, when

it comes to evaluation of the expressions of the integral. This issue is reflected in the mastery that went into Kahan's "atypically modest" expression in [7], which is written to optimize numerical stability of evaluation. We can, however, sidestep such concerns through the careful use of multiprecision numerics where the value is needed.

Theorem 2 *(Computable Forward Error) Given a rational function $f = A/B$ and tolerance ε, Algorithm 1 and Algorithm 2 yield an integral of a rational function $\hat{f}$ in the form (2) such that*

$$\|\int \Delta f \, dx\|_\infty = \max_x \left| \sum_k \left(\Xi(r_k, s_k, x) + \Theta(r_k, s_k, x)\right) \right| + O(\varepsilon^2),$$

where the leading term is $O(\varepsilon)$, r_k and s_k range over the real and imaginary parts of evaluated roots, and the functions Ξ and Θ defined below, corresponding to log and arctangent terms, respectively, are computable. This expression for the forward error is finite on any closed, bounded interval not containing a root of $B(x)$.

Proof (LRT-based forward error) Given the exact roots $c_{j\ell}$ of the $U_j(c)$, we can express the integral of the input rational function in the form

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \sum_{j=1}^{n} \sum_{\ell=1}^{\deg(U_j)} c_{j\ell} \cdot \log(S_j(c_{j\ell}, x)).$$

Since the roots $c_{j\ell}$ are complex, to get a real expression for the integral, we can convert the transcendental part into a sum of logarithms and arctangents using the real and imaginary parts of the $c_{j\ell}$.

For the remainder of the proof, we work with c_k, a subsequence of the roots $c_{j\ell}$ of the squarefree factors of the Rothstein-Trager resultant such that only one of each complex conjugate pair is included, and define φ to be a mapping given by $k \mapsto j$ so that $S_{\varphi(k)}(c_k, x)$ is the argument of the log term of the integral corresponding to the residue c_k. For each c_k, we let a_k and b_k be its real and imaginary parts, respectively. This allows us to express the integral in terms of logarithms and arctangent terms such that

$$\int f\,dx = \frac{C}{D} + \sum_{k=1}^{m} \left[a_k \log(V_k) + 2b_k \arctan(W_{1k}, W_{2k})\right], \tag{19}$$

where V_k, W_{1k}, and W_{2k} are functions of a_k, b_k, and x, and m is the size of the set $\{c_k\}$ of residues.

Once again, since the rational part of the integral is computed exactly, it does not contribute to the forward error. The forward error is the result of the evaluation of the

above expression at approximate values for the a_k and b_k. Therefore, considering the variation of (19) with respect to changes in the a_k and b_k, we obtain

$$\begin{aligned}\int \Delta f\, dx &= \int (f - \hat{f})(x)\, dx \\ &= \sum_{k=1}^{m} \left\{ \left[\left(\frac{\partial V_k}{\partial a_k} \Delta a_k + \frac{\partial V_k}{\partial b_k} \Delta b_k \right) \frac{a_k}{V_k} + \log(V_k) \Delta a_k \right] \right. \\ &+ \left[\left(W_{2k} \frac{\partial W_{1k}}{\partial a_k} - W_{1k} \frac{\partial W_{2k}}{\partial a_k} \right) \Delta a_k \right. \\ &+ \left. \left(W_{2k} \frac{\partial W_{1k}}{\partial b_k} - W_{1k} \frac{\partial W_{2k}}{\partial b_k} \right) \Delta b_k \right] \frac{2 b_k}{W_{1k}^2 + W_{2k}^2} \\ &+ \left. 2 \arctan(W_{1k}, W_{2k}) \Delta b_k \right\} + o(\Delta a_k, \Delta b_k). \end{aligned} \tag{20}$$

We now consider how to determine the values of V_k, W_{1k}, W_{2k} and their partial derivatives from information in the computed integral. To simplify notation, we let $j = \varphi(k)$. If c_k is real, then we obtain a term of the form $a_k \log(S_j(a_k, x))$. In the complex case, each c_k stands for a complex conjugate pair. As such, we obtain terms of the form

$$(a_k + i\, b_k) \log(S_j(a_k + i\, b_k, x)) + (a_k - i\, b_k) \log(S_j(a_k - i\, b_k, x).$$

Expressing $S_j(a_k + i\, b_k, x)$ in terms of real and imaginary parts as $W_{1k}(x) + i\, W_{2k}(x) \equiv W_{1k}(a_k, b_k, x) + i\, W_{2k}(a_k, b_k, x)$, so that $S_j(a_k - i\, b_k, x) = W_{1k}(x) - i\, W_{2k}(x)$, the expression of the term in the integral becomes

$$a_k \log\left(W_{1k}(x)^2 + W_{2k}(x)^2 \right) + i\, b_k \log\left(\frac{W_{1k}(x) + i\, W_{2k}(x)}{W_{1k}(x) - i\, W_{2k}(x)} \right).$$

The observation that $i \log\left(\frac{X+iY}{X-iY}\right)$ has the same derivative as $2 \arctan(X, Y)$ allows the term of the integral to be converted into the form of the summand in (19) with $V_k(x) = W_{1k}(x)^2 + W_{2k}(x)^2$.

To facilitate implementation, we can express V_k, W_{1k}, and W_{2k} and their partials in terms of $S_j(c, x)$ and $\partial S_j(c, x)/\partial c$ as follows. First of all, we have that

$$W_{1k}(x) = \mathsf{Re}(S_j(c_k, x)), \quad W_{2k}(x) = \mathsf{Im}(S_j(c_k, x)). \tag{21}$$

Then, because c is an indeterminate in $S_j(c, x)$, $\left.\frac{\partial S_j(c,x)}{\partial c}\right|_{c=c_k} = \frac{\partial S_j(c_k,x)}{\partial a_k}$ with $\frac{\partial S_j(c_k,x)}{\partial a_k} = \frac{\partial W_{1k}(c_k,x)}{\partial a_k} + i\, \frac{\partial W_{2k}(c_k,x)}{\partial a_k}$, so that

$$\frac{\partial W_{1k}}{\partial a_k} = \mathsf{Re}\left(\left.\frac{\partial S_j(c, x)}{\partial c}\right|_{c=c_k} \right), \quad \frac{\partial W_{2k}}{\partial a_k} = \mathsf{Im}\left(\left.\frac{\partial S_j(c, x)}{\partial c}\right|_{c=c_k} \right). \tag{22}$$

In a similar way, and because the derivative with respect to b_k picks up a factor of i, $\frac{\partial W_{1k}}{\partial b_k} = -\frac{\partial W_{2k}}{\partial a_k}$ and $\frac{\partial W_{2k}}{\partial b_k} = \frac{\partial W_{1k}}{\partial a_k}$. It follows, then, that $\frac{\partial V_k}{\partial a_k} = 2\left(W_{1k} \frac{\partial W_{1k}}{\partial a_k} + W_{2k} \frac{\partial W_{2k}}{\partial a_k} \right)$ and $\frac{\partial V_k}{\partial b_k} = 2\left(W_{2k} \frac{\partial W_{1k}}{\partial a_k} - W_{1k} \frac{\partial W_{2k}}{\partial a_k} \right)$.

For the complex root case, given the error bound $|\Delta c| \le \varepsilon |\hat{c}|$ on the complex roots, we have the same bound on the real and imaginary parts, viz., $|\Delta a| \le \varepsilon |\hat{a}|$,

$|\Delta b| \le \varepsilon|\hat{b}|$. Since $\Delta c_k = -U_j(\hat{c}_k)/U_j'(\hat{c}_k)$ to first order, and $\Delta c_k = \Delta a_k + i\,\Delta b_k$, from (20) we therefore obtain an expression for the linear forward error in the form required by the theorem with

$$\boxed{\Xi(\hat{a}_k, \hat{b}_k, x) = \left(2\hat{a}_k\Gamma + \log\left(W_{1k}^2 + W_{2k}^2\right)\right)\mathsf{Re}\left(\frac{U_j}{U_j'}\right) + 2\hat{a}_k\Lambda\,\mathsf{Im}\left(\frac{U_j}{U_j'}\right)}$$

when $a_k \neq 0$, otherwise $\Xi(\hat{a}_k, \hat{b}_k, x) \equiv 0$, and with

$$\boxed{\Theta(\hat{a}_k, \hat{b}_k, x) = 2\hat{b}_k\Lambda\,\mathsf{Re}\left(\frac{U_j}{U_j'}\right) + 2\left(\operatorname{artcan}(W_{1k}, W_{2k}) - \hat{b}_k\Gamma\right)\mathsf{Im}\left(\frac{U_j}{U_j'}\right)},$$

where $\Gamma = \frac{W_{1k}\frac{\partial W_{1k}}{\partial a_k} + W_{2k}\frac{\partial W_{2k}}{\partial a_k}}{W_{1k}^2 + W_{2k}^2}$, $\Lambda = \frac{W_{2k}\frac{\partial W_{1k}}{\partial a_k} - W_{1k}\frac{\partial W_{2k}}{\partial a_k}}{W_{1k}^2 + W_{2k}^2}$, W_{1k} and W_{2k} are given by (21), $\frac{\partial W_{1k}}{\partial a_k}$ and $\frac{\partial W_{2k}}{\partial a_k}$ are given by (22), and all expressions, including U_j and U_j', are evaluated at $\hat{c}_k = \hat{a}_k + i\,\hat{b}_k$. These terms are $O(\varepsilon)$ because $\frac{U_j(\hat{c}_k)}{U_j'(\hat{c}_k)}$ is $O(\varepsilon)$.

For the real root case, we have a much simpler expression, since $\Theta(\hat{a}_k, \hat{b}_k, x) \equiv 0$ and since $\hat{c}_k = \hat{a}_k$,

$$\boxed{\Xi(\hat{a}_k, \hat{b}_k, x) = \left(\hat{a}_k \frac{\left.\frac{\partial S_j}{\partial c}\right|_{c=\hat{a}_k}}{S_j(\hat{a}_k, x)} + \log(S_j(\hat{a}_k, x))\right)\frac{U_j(\hat{\alpha}_k)}{U_j'(\hat{\alpha}_k)}},$$

which is also $O(\varepsilon)$. □

Proof (PFD-based forward error) Proceeding as we did for the LRT method, if we assume that the roots of the denominator of the polynomial $H(x)$ are computed exactly, then we obtain an exact expression of the integral of f in the form

$$\int f(x)\,dx = \frac{C(x)}{D(x)} + \sum_{i=1}^{\deg(H)} c_i(\gamma_i)\log(x - \gamma_i). \tag{23}$$

As in the LRT-based proof, we assume γ_k is a subsequence of the γ_i that includes only one conjugate of each complex root. Then the same techniques for converting this to a sum of logarithms and arctangents can be applied here. Since $H(x)$ is squarefree, all of the $\gamma_k = \alpha_k + i\,\beta_k$ are simple roots, which entails that the integral can be expressed in the form (19) where the $V_j(x)$ are equal to $x - \alpha_k$ for a real root and $x^2 - 2\alpha_k + \alpha_k^2 + \beta_k^2$ for a complex root with $a_k = \mathsf{Re}(c(\gamma_k))$ where $c(x) = G(x)/H'(x)$. Using the RSR method, $W_{1k}(x) = \alpha_k - x$, $W_{2k} = \beta_k$ and $b_k = \mathsf{Im}(c(\gamma_k))$. Even though the structured integral is not expressed in this form, it is still an exact integral that we approximate, where all subsequent computation we perform is exact. Analyzing the error in this form has the advantage of using information available immediately after the completion of the rootfinding task. Thus, we will analyze the forward error in this form.

Because the residues are now obtained by computation, and we compute the roots $\gamma_k = \alpha_k + i\,\beta_k$ of $H(x)$, we obtain a modified version of the first-order forward error formula (20), viz.,

$$\begin{aligned}\int \Delta f\,dx &= \int (f - \hat{f})(x)\,dx \\ &= \sum_{k=1}^{m} \left\{ \left[\left(\frac{\partial V_k}{\partial \alpha_k}\Delta\alpha_k + \frac{\partial V_k}{\partial \beta_k}\Delta\beta_k \right) \frac{a_k}{V_k} + \left(\frac{\partial a_k}{\partial \alpha_k}\Delta\alpha_k + \frac{\partial a_k}{\partial \beta_k}\Delta\beta_k \right) \log(V_k) \right] \right. \\ &\quad \left. + \frac{2\beta_k b_k(\Delta\alpha_k + \Delta\beta_k)}{(\alpha_k - x)^2 + \beta_k^2} + 2\left(\frac{\partial b_k}{\partial \alpha_k}\Delta\alpha_k + \frac{\partial b_k}{\partial \beta_k}\Delta\beta_k \right) \arctan(\alpha_k - x,\, \beta_k) \right\} \\ &\quad + o(\Delta\alpha_k,\, \Delta\beta_k). \end{aligned} \tag{24}$$

Since $c(x) = G(x)/H'(x)$, $c'(x) = \frac{G'(x)}{H'(x)} - \frac{G(x)H''(x)}{H'(x)^2}$, and so it follows that $\frac{\partial a_k}{\partial \alpha_k} = \mathsf{Re}(c'(\gamma_k))$ and $\frac{\partial b_k}{\partial \alpha_k} = \mathsf{Im}(c'(\gamma_k))$. Similarly, $\frac{\partial a_k}{\partial \beta_k} = -\mathsf{Im}(c'(\gamma_k))$ and $\frac{\partial b_k}{\partial \beta_k} = \mathsf{Re}(c'(\gamma_k))$. For the complex root case, then, since $\Delta\gamma_j = -H(\hat{\gamma}_j)/H'(\hat{\gamma}_j)$ to first order, $\Delta\alpha_j = -\mathsf{Re}(H(\hat{\gamma}_j)/H'(\hat{\gamma}_j))$ and $\Delta\beta_j = -\mathsf{Im}(H(\hat{\gamma}_j)/H'(\hat{\gamma}_j))$. Collecting terms with $\Delta\alpha_k$ together and terms with $\Delta\beta_k$ together, we obtain from (24) an expression for the linear forward error in the form required by the theorem with $\Xi(\hat{\alpha}_k, \hat{\beta}_k, x) = \Xi_\alpha + \Xi_\beta$, where

$$\boxed{\Xi_\alpha = \left(\frac{2\hat{a}_k(\hat{\alpha}_k - 1)}{(\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2} + \mathsf{Re}\left(c'(\hat{\gamma}_k)\right) \log\left((\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2 \right) \right) \mathsf{Re}\left(\frac{H}{H'} \right)}$$

and

$$\boxed{\Xi_\beta = \left(\frac{2\hat{a}_k\hat{\beta}_k}{(\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2} + \mathsf{Im}\left(c'(\hat{\gamma}_k)\right) \log\left((\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2 \right) \right) \mathsf{Im}\left(\frac{H}{H'} \right)}$$

when $\alpha_k \neq 0$, otherwise $\Xi(\hat{\alpha}_k, \hat{\beta}_k, x) \equiv 0$, and with $\Theta(\hat{\alpha}_k, \hat{\beta}_k, x) = \Theta_\alpha + \Theta_\beta$, where

$$\boxed{\Theta_\alpha = \left(\frac{2\hat{\beta}_k\hat{b}_k}{(\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2} - \mathsf{Im}\left(c'(\hat{\gamma}_k)\right) \arctan(\hat{\alpha}_k - x, \hat{\beta}_k) \right) \mathsf{Re}\left(\frac{H}{H'} \right)}$$

and

$$\boxed{\Theta_\beta = \left(\frac{2\hat{\beta}_k\hat{b}_k}{(\hat{\alpha}_k - x)^2 + \hat{\beta}_k^2} + \mathsf{Re}\left(c'(\hat{\gamma}_k)\right) \arctan(\hat{\alpha}_k - x, \hat{\beta}_k) \right) \mathsf{Im}\left(\frac{H}{H'} \right)},$$

with H and H' being evaluated in all cases at $\hat{\gamma}_k$. All of these terms are $O(\varepsilon)$ because $\frac{H}{H'}$ is.

In the case of real roots,

$$\boxed{\Xi(\hat{\alpha}_k, \hat{\beta}_k, x) = \left(c'(\hat{\alpha}_k)\log\left(x - \hat{\alpha}_k\right) - \frac{\hat{a}_k}{x - \hat{\alpha}_k}\right)\frac{H(\hat{\alpha}_k)}{H'(\hat{\alpha}_k)}},$$

which is also $O(\varepsilon)$. □

We note again that the forward error is structured for both algorithms. In the LRT-based case, the exact integral is computed and the approximation only perturbs the values of coefficients of polynomials in the integral, such that the (root and residue) multiplicity structure of the PFD of the derivative is preserved. In the PFD-based case, only the residue structure of the PFD of the derivative of the transcendental part is preserved due to the identification of the residues that are within ε of each other,[5] though the integral of the rational part is exact. Thus, the PFD-based method achieves a result only approximately on the pejorative manifold, while the LRT-based method keeps the result exactly on the pejorative manifold.[6]

Once again, note that the scaling behaviour for the error term for real roots can be used to efficiently compute the boundaries around the singularities in the integral. In this case, the error scales as $(x - \alpha)^{-1}$ and $S_j(a_k, x)^{-1}$, since the quadratic terms appearing the backward error have been integrated. As a result, the forward error grows much more slowly than the backward error does as we approach a singularity, and we get much smaller bounds before the error exceed the tolerance.

5 Implementation

We have implemented the algorithms presented in Section 3. In our code, the symbolic computations are realized with the *Basic Polynomial Algebra Subprograms* (BPAS) publicly available in source at http://bpaslib.org. The BPAS library offers polynomial arithmetic operations (multiplication, division, root isolation, etc.) for univariate and multivariate polynomials with integer, rational or complex rational number coefficients, as well as exact solution of non-linear systems for polynomials with rational number coefficients; it is written in C and C++ with the CilkPlus extension for optimization on multicore architectures and is built on top of the GMP library and MPSOLVE, which we now describe.

[5]This means that when several computed residues are coalesced, the chosen representative root $\hat{\gamma}_{\bar{k}}$ must be used to evaluate $\hat{a}_k$ and $\hat{b}_k$, and $c'(\hat{\gamma}_k)$ must be used to evaluate all of the error terms corresponding to the roots $\hat{\gamma}_k$ that have the same residue.

[6]Note that this may account for the differences in stability between the LRT-based and PFD-based methods observed in Section 6 below. Because roots are isolated using multiprecision arithmetic, however, both methods nonetheless produce stable results.

The numerical portion of our computation relies on MPSOLVE, publicly available in source at http://numpi.dm.unipi.it/mpsolve. The MPSOLVE library, which is written in C and built upon the GMP library, offers arbitrary precision solvers for polynomials and secular equations of the form $S(x) = 0$ with $S(x) = \sum_{i=1}^{n} \frac{a_i}{x-b_i} - 1$, with a posteriori guaranteed inclusion radii, even on restricted domains, such as the real line. For a requested output precision of 2^{-d}, it can ensure at least d correct digits in the returned roots (see [1] for more details).

The implementation of both Algorithm 1 and Algorithm 2 are integrated into the BPAS library; Algorithms 1 and 2 can be called, respectively, through the `realSymbolicNumericIntegrate` and `realSymbolicNumericIntegratePFD` methods of the `UnivariateRationalFunction` template class. We abbreviate the `realSymbolicNumericIntegrate` method to `snIntLRT` and the `realSymbolicNumericIntegratePFD` method as `snIntPFD` in the sequel. The following output formats are available in the `UnivariateRationalFunction` class: approximate (floating point number) or symbolic (rational number) expressions (in either MAPLE or MATLAB syntax); see Fig. 2 for a combination of floating point and MAPLE output formats.

For the integral $\int \frac{(x^2-1)dx}{x^4+5x^2+7}$, MAPLE provides the expression appearing in Fig. 1. For the same integral, the BPAS/MPSOLVE routines `snIntLRT` and `snIntPFD` both return the output shown in Fig. 2 in the default floating point output format. In the data structures, however, the coefficients are stored as multiprecision rational numbers, which can be displayed by changing the output format.

It must be noted that there are differences between Algorithms 1 and 2 and their implementations in BPAS. The key difference is that `snIntPFD` and `snIntLRT` do additional post-processing. As such, the forward error analysis detailed in Section 4 assumes a different output expression than is produced finally in the implementations. Both `snIntPFD` and `snIntLRT` do, however, compute the integral in the form assumed in the forward error analysis. There are therefore several reasons why the additional post-processing will not significantly affect the conclusions drawn from the error analysis.

First of all, after the integral is computed in the form of Eq. (19), all further computation in BPAS is done using exact computation. As such, the final expression, which uses the RSR method to remove spurious singularities from the arctangents, is mathematically equivalent to the integral in the form of (19) from the perspective of the integration problem, viz., they have the same derivative and hence differ only by a constant.

```
  /
 /   -1+x^2
 | ----------- dx =
 / 7+5*x^2+x^4
 /
 -0.638053*ln(2.64575+0.53991*x+x^2) + 0.638053*ln(2.64575-0.53991*x+x^2) + 0
.0969495*arctan(0.168299+0.623434*x) + 0.0969495*arctan(-0.168299+0.623434*x)
```

Fig. 2 Sample output of `snIntLRT` and `snIntPFD`

Another reason why the additional post-processing will not affect the forward error evaluation is that converting two-argument arctangent functions of polynomials (or one-argument arctangents of rational functions) to one-arguments arctangents of polynomials increases their numerical stability. This is because the derivative of $\arctan(x)$ is $1/(1+x^2)$, which can never be zero, or even small, for real integrals, whereas the derivative of $\arctan(x_1, x_2)$ and $\arctan(x_1/x_2)$ is $(x_1' x_2 - x_1 x_2')/(x_1^2 + x_2^2)$, which can approach zero for nearly real roots of the denominator of the integrand. This changes the denominators of the expressions for $\Theta(r_k, s_k, x)$ appearing in the proof of Theorem 2. Thus, the application of the RSR method improves the stability of the integral. As such, the worst that can happen in this situation is that the forward error appears large from the perspective of the error analysis when it actually is not. Though this issue will need to be resolved in refinements of the implementation, it is very unlikely to be a significant issue on account of the fact that the error is dominated by the error in the roots, and in practice this error is many orders of magnitude less than the tolerance.

Since the forward error analysis is reliable, modulo the issue just stated, even though we could compute the forward error on the final output, it is a design decision not to do so. This is because a design goal of the algorithm is to hide the error analysis in the main computation of the integral by performing the error analysis in parallel. This is only possible if the error analysis can proceed on information available before the integral computation is completed.

6 Experimentation

We now consider the performance of Algorithms 1 and 2 based on their implementations in BPAS.[7] For the purposes of comparing their runtime performance we will consider integration of the following functions:

1. $f_1(x) = \frac{1}{x^n - 2}$;
2. $f_2(x) = \frac{1}{x^n + x - 2}$;
3. $f_3(x) = [n, n]_{e^x/x}(x)$,

where $[m, n]_f(x)$ denotes the Padé approximant of order $[m/n]$ of f. Since $\int \frac{e^x}{x} dx = \mathrm{Ei}(x)$, the non-elementary exponential integral, integrating f_3 provides a way of approximating $\mathrm{Ei}(x)$. These three problems test different features of the integrator on account of the fact that $f_1(x)$ has a high degree of symmetry, while $f_2(x)$ breaks this symmetry, and $f_3(x)$ contains very large integer coefficients for moderate size n. Note that unless otherwise stated, we are running `snIntPFD` and `snIntLRT` with the error analysis computation turned on.

Comparing `snIntPFD` and `snIntLRT` on functions f_1 and f_2 for 40 logarithmically spaced values of n from $n = 8$ to $n = 377$, we find the results shown in Fig. 3. We see from Fig. 3a that the performance of the two algorithms is nearly identical on function $f_1(x)$. Figure 3b shows, however, that on function $f_2(x)$,

[7] The data for this section was collected using an internal December 2017 version of the BPAS library.

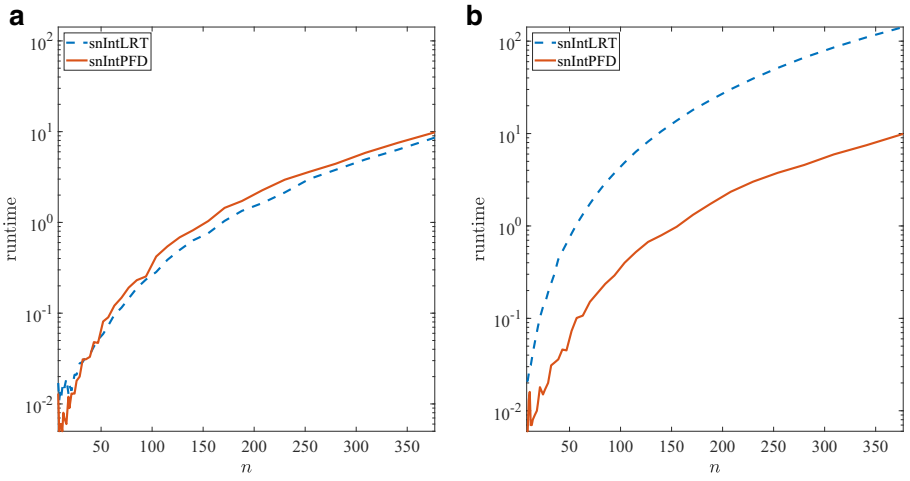

Fig. 3 Runtime comparison of `snIntPFD` and `snIntLRT` on problems **a** $f_1(x)$ and **b** $f_2(x)$

`snIntLRT` performs considerably poorer than `snIntPFD`. The reason for this is that the size of the coefficients in the Rothstein-Trager resultant grows exponentially for $f_2(x)$. This causes no significant issues for the subresultant computation, but it significantly slows the rootfinding algorithm, leading to large rational number roots, which slows the post-processing algorithms. In contrast, the difference in runtime for `snIntPFD` on functions $f_1(x)$ and $f_2(x)$ is negligible. This is because the speed of `snIntPFD` is determined by the degree of the denominator (after the squarefree part has been extracted by Hermite reduction) and the height of the coefficients. Since the denominators of f_1 and f_2 have the same degree and height bound, we expect the performance to be similar.

If we run the same computation with the error analysis turned off, we would see that `snIntLRT` actually performs better than `snIntPFD` on problem 1. With the relative performance improvement of `snIntLRT` relative to `snIntPFD` being similar to the performance of `snIntPFD` relative to `snIntLRT` on problem 2 with the error analysis turned on. Thus, there are some problems on which `snIntLRT` performs better than `snIntPFD`. The performance of `snIntPFD` is easier to predict from the degree and height bound of the input polynomials.

That there is a difference in performance in `snIntLRT` when the error analysis computation is turned off shows that the current implementation can only partially hide the error analysis computation for `snIntLRT` on some problems. The error analysis computation is successfully completely hidden for problem 2 with `snIntLRT`, which is to be expected. For `snIntPFD`, however, there is a negligible difference in the runtime with the error analysis turned on and off. Thus, once again, `snIntPFD` has the more reliable and desirable behaviour.

`snIntPFD` also performs better on problem 3, which leads to coefficients with height bound, measured as an absolute value, that grows exponentially with n. For $n = 8$, `snIntLRT` computes the integral in about 0.04 s, whereas `snIntPFD` computes it in about 0.01 s. For $n = 13$, the respective runtimes increase to 0.18 s and

Table 1 Tolerance proportionality of the global forward and backward error bounds for `snIntLRT` and `snIntPFD` on $\int \frac{dx}{x^{128}+2}$

	`snIntLRT`		`snIntPFD`	
ε	Forward error	Backward error	Forward error	Backward error
$6 \cdot 10^{-11}$ (2^{-34})	$8 \cdot 10^{-16}$	$1 \cdot 10^{-15}$	$2 \cdot 10^{-15}$	$2 \cdot 10^{-12}$
$3 \cdot 10^{-17}$ (2^{-55})	$3 \cdot 10^{-55}$	$2 \cdot 10^{-53}$	$1 \cdot 10^{-39}$	$1 \cdot 10^{-38}$
$2 \cdot 10^{-27}$ (2^{-89})	$1 \cdot 10^{-75}$	$2 \cdot 10^{-73}$	$9 \cdot 10^{-59}$	$8 \cdot 10^{-58}$
$4 \cdot 10^{-44}$ (2^{-144})	$6 \cdot 10^{-95}$	$7 \cdot 10^{-93}$	$3 \cdot 10^{-77}$	$2 \cdot 10^{-76}$

0.02 s, and by $n = 21$, around 2.5 s and 0.04 s. This shows that `snIntLRT` is considerably slowed down by large input coefficients, since this leads to even larger coefficients in the subresultants. This is reflected in the subresultant computation taking 0.6 s for $n = 21$ and slowing down the exact integration to 2.4 s. Thus, when it comes to runtime performance, `snIntPFD` is the clear winner.

Turning to the error analysis, we now consider the behaviour of the error under variation of the input tolerance ε. For integrands without real singularities, we can compute a global forward and backward error bound over the entire real line. For the non-singular problem $\int \frac{dx}{x^{128}+2}$, a variant of problem 1, we see from Table 1 that both `snIntLRT` and `snIntPFD` exhibit tolerance proportionality as the tolerance is reduced. Here `snIntLRT` generally outperforms `snIntPFD` for a given input tolerance by several orders of magnitude, but both algorithms perform strongly.

On problems that do have real singularities, we obtain boundaries around the singularities past which the error exceeds the input tolerance. On problem 3 for $n = 8$, there is a real singularity at $x \doteq 10.949$. For this singularity, we see from Table 2 that both `snIntLRT` and `snIntPFD` exhibit tolerance proportionality of the singularity boundaries as the tolerance is reduced. Thus, we can get as close to the singularity as desired by decreasing the input tolerance. With the exception of $\varepsilon = 2^{-34}$, `snIntLRT` outperforms `snIntPFD`, but the difference in performance between the two algorithms is not as extreme as with the non-singular case. For the default

Table 2 Tolerance proportionality of the singularity boundary widths for `snIntLRT` and `snIntPFD` on problem 3 with $n = 8$ for the singularity at $x \doteq 10.949$

	`snIntLRT`		`snIntPFD`	
ε	Forward error ∂	Backward error ∂	Forward error ∂	Backward error ∂
2^{-34}	$4 \cdot 10^{-3}$	$6 \cdot 10^{-2}$	$1 \cdot 10^{-14}$	$6 \cdot 10^{-7}$
2^{-55}	$7 \cdot 10^{-23}$	$9 \cdot 10^{-12}$	$8 \cdot 10^{-20}$	$3 \cdot 10^{-10}$
2^{-89}	$4 \cdot 10^{-32}$	$2 \cdot 10^{-16}$	$2 \cdot 10^{-28}$	$1 \cdot 10^{-14}$
2^{-144}	$2 \cdot 10^{-34}$	$1 \cdot 10^{-17}$	$3 \cdot 10^{-31}$	$6 \cdot 10^{-16}$

The symbol ∂ is used to abbreviate "boundary width"

precision of $\varepsilon = 2^{-53}$ and above, both algorithms get extremely close to the singularity before the error exceeds the tolerance.

For testing the numerical stability, we will consider two additional problems, along with problem 3 above:

4. $f(x) = \frac{2x}{x^2-(1+\epsilon)^2}, \epsilon \to 0$ (singular just outside $[-1, 1]$);
5. $f(x) = \frac{2x}{x^2+\epsilon^2}, \epsilon \to 0$ (nearly real singularities on the imaginary axis).

Note that the small parameter ϵ in problems 4 and 5 is conceptually distinct from the input tolerance ε. These problems are useful for testing the stability of the integration algorithms near singularities.

On problems 4 and 5, `snIntLRT` computes the exact integral, because the integral contains only rational numbers, so there is no need to do any rootfinding. Thus, the forward and backward errors are exactly zero, and the evaluation of the integral is insensitive to how close the singularities are to the boundary of the interval of integration, provided a numerically stable method of evaluating the logarithm is used.

On the same problems `snIntPFD` computes very nearly the exact integral. On problem 4, with $\varepsilon = 2^{-53}$, it is possible to get to within about $1.6 \cdot 10^{-23}$ of the singularities at $\pm 1 \pm \epsilon$ before the error exceeds the tolerance. Thus, even with $\epsilon = \varepsilon$, the error does not affect the evaluation of the integral on the interval $[-1, 1]$. `snIntPFD` also performs exceedingly well on problem 5. With the same input tolerance, the forward error bound is $1.9 \cdot 10^{-57}$ for $\epsilon = 0.1$ and increases only to $1.7 \cdot 10^{-42}$ for $\epsilon = 10^{-16}$. Indeed, the difference between the a in $\log(x^2 + a)$ computed by `snIntLRT` and `snIntPFD` is about $1.7 \cdot 10^{-74}$.

Since problem 3 requires rootfinding for both algorithms, it provides a more fair comparison. `snIntLRT` fares slightly better than `snIntPFD` on this problem, but only slightly and not in a way that significantly affects numerical evaluation. We make the comparison for $\varepsilon = 2^{-53}$. With $n = 8$, for `snIntLRT` the backward and forward error bounds away from the real singularities are about $2.1 \cdot 10^{-38}$ and $1.8 \cdot 10^{-36}$, respectively. For `snIntPFD` the backward error bound increases only to $2.6 \cdot 10^{-35}$ and the forward error bound decreases slightly to $1.2 \cdot 10^{-36}$. As for evaluation near the real singularities, `snIntLRT` can get within about $1.8 \cdot 10^{-23}$ of the singularity at around $x = 10.949$ before the forward error exceeds the tolerance. `snIntPFD` can get within about $2 \cdot 10^{-20}$. Of course, this is not a true concern anyway, because the Padé approximant ceases to be a good approximation of e^x/x before reaching the real root.

We see, therefore, that `snIntPFD` performs strongly against `snIntLRT` even when `snIntLRT` gets the exact answer and `snIntPFD` does not. Indeed, the differences in numerical stability between the two methods are relatively small. Given the performance benefits of `snIntPFD`, the PFD-based algorithm is the clear overall winner between the two algorithms. `snIntPFD` is therefore the preferred choice, except in cases where the exact integral needs to be retained.

7 Conclusion and future work

We have identified two methods for the hybrid symbolic-numeric integration of rational functions on exact input that adjust the forward and backward errors of

the integration according to a user-specified tolerance, determining a posteriori the intervals of integration on which the integration is numerically stable. We provided computable expressions for the first-order term of the forward and backward error for both integration methods. The PFD-based method is overall the better algorithm, being better overall in terms of runtime performance while maintaining excellent numerical stability. The LRT-based method is still advantageous in contexts where the exact integral needs to be retained for further symbolic computation. We believe these algorithms, and the extension of this approach to wider classes of integrands, has potential to increase the utility of symbolic computation in scientific computing.

In footnote 4 of Section 4, we pointed out difficult special cases that can arise when handling coalescing of residues in the PFD-based algorithm for special cases of nearby residues. To illustrate the kind of issue here, consider the case where, for input tolerance ε, we want to compute $\int g(x)dx = \int \frac{G(x)}{H(x)}dx$, with $H(x) = (x-\alpha)(x-\beta)(x-\gamma)(x-\delta)$, such that

$$g(x) = \frac{1-3\varepsilon/2}{x-\alpha} + \frac{1-\varepsilon/2}{x-\beta} + \frac{1+\varepsilon/2}{x-\gamma} + \frac{1+3\varepsilon/2}{x-\delta}. \tag{25}$$

Thus, the exact residues are all ε apart, and all residues are within 3ε of each other. This gives rise to a decision of how, and whether, to coalesce residues. Assume, for simplicity of presentation, that we compute the residues exactly but still wish to coalasce nearby residues for simpler expressions where possible. If, for example, we identified the first two and last two residues in (25), we could obtain an expression for the integral such as

$$\int g(x)dx \doteq (1-\varepsilon)\log((x-\alpha)(x-\beta)) + (1+\varepsilon)\log((x-\gamma)(x-\delta)),$$

while identifying all the residues would yield an expression such as

$$\int g(x)dx \doteq \log((x-\alpha)(x-\beta)(x-\gamma)(x-\delta)).$$

The latter might be desirable from the perspective of the simpler coefficient of the logarithm and its having only a single logarithm term, but there are other concerns here. The important thing to notice is that these two expressions behave very differently from the perspective of the numerical evaluation of the integral on account of the ill-conditioning of evaluation of the logarithm near 1, which happens in different places for the different polynomial arguments of the different logarithm terms. It is therefore not obvious which expression to choose. In the context of multiprecision evaluation of the integral this issue is mitigated, but then the solution to this issue becomes implementation dependent. In future work we intend to treat this issue more thoroughly to develop a more robust version of the algorithms presented here.

References

1. Bini, D.A., Robol, L.: Solving secular and polynomial equations: a multiprecision algorithm. J. Comput. Appl. Math. **272**, 276–292 (2014)
2. Bronstein, M.: Symbolic Integration, vol. 3. Springer, Berlin (1997)

3. Bronstein, M.: Symbolic integration tutorial. https://www-sop.inria.fr/cafe/Manuel.Bronstein/publications/issac98.pdf. Accessed: 2019-04-26 (1998)
4. Buchberger, B., Rüdiger, G.K.L.: Algebraic Simplification, pp. 11–43. Springer, Wien–New York (1982)
5. Fateman, R.: Revisiting numeric/symbolic indefinite integration of rational functions, and extensions. https://people.eecs.berkeley.edu/fateman/papers/integ.pdf. Accessed: 2019-04-26 (2008)
6. Kahan, W.M.: Conserving confluence curbs ill-condition. Technical report, DTIC Document (1972)
7. Kahan, W.M.: Handheld calculator evaluates integrals. Hewlett-Packard Journal **31**(8), 23–32 (1980)
8. Noda, M.-T., Miyahiro, E.: On the symbolic/numeric hybrid integration. In: Proceedings of the International Symposium on Symbolic and Algebraic Computation, p. 304. ACM (1990)
9. Noda, M.-T., Miyahiro, E.: A hybrid approach for the integration of a rational function. J. Comput. Appl. Math. **40**(3), 259–268 (1992)
10. Rioboo, R.: Proceedings of the 1992 International Symposium on Symbolic and Algebraic Computation, ISSAC '92, Berkeley, CA, USA, July 27–29, 1992. In: Wang, P.S. (ed.), pp. 206–215. ACM (1992)
11. Rioboo, R.: Towards faster real algebraic numbers. J. Symb. Comput. **36**(3–4), 513–533 (2003)
12. Zeng, Z.: Apatools: a software toolbox for approximate polynomial algebra. In: Software for Algebraic Geometry, pp. 149–167. Springer (2008)