Numerical Integration with Exact Real Arithmetic*

Abbas Edalat and Marko Krznarić

Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, UK
ae@doc.ic.ac.uk, marko@doc.ic.ac.uk

Abstract. We show that the classical techniques in numerical integration (namely the Darboux sums method, the compound trapezoidal and Simpson's rules and the Gauss–Legendre formulae) can be implemented in an exact real arithmetic framework in which the numerical value of an integral of an elementary function is obtained up to any desired accuracy without any round–off errors. Any exact framework which provides a library of algorithms for computing elementary functions with an arbitrary accuracy is suitable for such an implementation; we have used an exact real arithmetic framework based on linear fractional transformations and have thereby implemented these numerical integration techniques. We also show that Euler's and Runge–Kutta methods for solving the initial value problem of an ordinary differential equation can be implemented using an exact framework which will guarantee the convergence of the approximation to the actual solution of the differential equation as the step size in the partition of the interval in question tends to zero.

1 Introduction

In numerical methods for integration of a real–valued function, such as in the Darboux sums method, the trapezoidal method, Simpson's rule or the Gauss–Legendre formulae, the value of an integral $\mathrm{I}f = \int_a^b f(x)dx$ is approximated by a sum $\mathrm{Q}_n f = \sum_{i=0}^n c_i f(x_i)$, where the x_i's are the integration nodes and the c_i's are the corresponding integration weights. In each method, an upper bound for the error can be obtained in terms of the number n of the integration nodes and an upper bound for a higher derivative of the integrand. Mathematically, this error always tends to zero as n tends to infinity. However, when any of these methods is implemented in floating point, one obtains an extra round–off error in the computation of $\mathrm{Q}_n f$ and, as n tends to infinity, the computed values of $\mathrm{Q}_n f$ may fail to converge to the value of the integral [4]. This same problem occurs in the floating point implementation of the Euler's and Runge–Kutta methods for the numerical computation of the solution of an initial value problem for an ordinary differential equation.

This undesirable situation can be overcome in exact real arithmetic. Simpson [21] has developed a theoretical setting for the exact computation of integrals

* This work is supported by EPSRC.

using higher order functions. However, the resulting algorithms are intractable. This is even more so in the earlier work of Edalat and Escardó [7] which defines integration in an extension of the theoretical language PCF.

We show here that one can use any of the above methods of integration to obtain the numerical value of the integral of any elementary function up to any desired accuracy ϵ, provided that we have: (i) a set of algorithms for the computation of elementary functions with arbitrary accuracy, and, (ii) an upper bound for the corresponding higher derivative of the integrand. This is achieved by first writing $\epsilon = \epsilon_m + \epsilon_r$ with $\epsilon_m, \epsilon_r > 0$. Then, for a given integration method, we determine the number of nodes such that $\mathrm{Q}_n f$ differs from the integral by at most ϵ_m; finally $\mathrm{Q}_n f$ is evaluated using the exact real arithmetic framework up to precision ϵ_r, which means the result differs from the integral by at most ϵ as desired. We also show that by using such an exact framework, one can obtain a convergent sequence of approximations with the Euler's and Runge–Kutta methods as the number of nodes tends to infinity.

We actually use the exact real arithmetic framework based on linear fractional transformations (LFTs) as developed by Edalat and Potts [8, 6, 20] based on the work of Gosper [12], Vuillemin [22] and Kornerup and Nielsen [19] on the one hand and Di Gianantonio [5] and Escardó [11] on the other. Using the complexity results of Heckmann [13] for computing elementary functions in the LFT framework, we obtain, as an example, the complexity of integrating the exponential function in the Darboux sums method, the trapezoidal method and in Simpson's rule in this framework. A number of examples for numerical integration using the above different methods and for the initial value problem using the Euler's and Runge–Kutta methods are presented.

2 Numerical Integration

We will deal with the numerical integration of a real–valued bounded function f over a finite closed interval $[a, b]$:

$$\mathrm{I}f = \int_a^b f(x)dx.$$

The simplest way to approximate the value $\mathrm{I}f$ is by a linear combination $\mathrm{Q}_n f$ of the values of f:

$$\begin{aligned} \mathrm{I}f &\approx \mathrm{Q}_n f \\ &= c_0 f(x_0) + c_1 f(x_1) + \ldots + c_n f(x_n). \end{aligned}$$

Some integration methods are employed on specific intervals (e.g. Gauss–Legendre formula is given for the interval $[-1, 1]$). In order to apply such methods on another interval, we have to transform an integral over an arbitrary interval $[a, b]$ to an integral over $[-1, 1]$:

$$\int_a^b f(x)dx = \frac{b-a}{2} \int_{-1}^1 f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right) dx.$$

We summarize four integration formulae; more details about these methods can be found in any book about numerical integration (see, for example [16, 4]):

Darboux sums For a bounded function $f : [a, b] \to \mathbb{R}$, and an equipartition $\mathcal{P} = \{x_0, x_1, \ldots, x_n\}$ of $[a, b]$, the lower Darboux sum is given by $\underline{S}(f; \mathcal{P}) = \frac{b-a}{n} \sum_{k=1}^{n} m_k$, where m_k is the minimum value of f on the subinterval $[x_{k-1}, x_k]$. If additionally f is a monotone function, then the minimum m_k is obtained at one of the end–points of the subinterval $[x_{k-1}, x_k]$, while n can be obtained as the least integer n which satisfies $\frac{b-a}{\epsilon}|f(b) - f(a)| < n$, where $\epsilon > 0$ is the required accuracy.

Closed Newton–Cotes formulae The compound trapezoidal and Simpson's rules are constructed by dividing $[a, b]$ into n subintervals by an equipartition $\mathcal{P} = \{x_0, x_1, \ldots, x_n\}$ and approximating f on each subinterval by polynomials of degree one and two respectively. These two rules are respectively given by:

$$\mathrm{Q}_1^n f = \frac{b-a}{2n} \left[f(x_0) + 2 \sum_{k=1}^{n-1} f(x_k) + f(x_n) \right], \tag{1}$$

$$\mathrm{Q}_2^n f = \frac{b-a}{6n} \left[f(x_0) + 2 \sum_{k=1}^{n-1} f(x_k) + 4 \sum_{k=0}^{n-1} f(x_{k+\frac{1}{2}}) + f(x_n) \right], \tag{2}$$

where $x_{k+\frac{1}{2}} = (x_k + x_{k+1})/2$. We may obtain an approximation which differs from $\mathrm{I}f$ by at most $\epsilon > 0$, provided that f is smooth enough ($f \in C^2[a, b]$ for the trapezoidal rule, and $f \in C^4[a, b]$ for Simpson's rule). This is done by taking $\frac{B(b-a)^3}{12\epsilon} < n^2$ and $\frac{C(b-a)^5}{2880\epsilon} < n^4$ for the trapezoidal and Simpson's rules respectively. B and C are upper bounds for $|f^{(2)}|$ and $|f^{(4)}|$ respectively on the interval $[a, b]$. For given n, the number of integration nodes for the trapezoidal rule is $(n + 1)$, while for Simpson's rule is $(2n + 1)$.

Gauss–Legendre formulae In the formulae above, the choice of the integration nodes (equipartition of $[a, b]$), does not guarantee the maximum possible degree of accuracy[1]. The Gauss formulae, however, guarantee this accuracy. Here, we discuss the Gauss–Legendre formula, with the roots of Legendre polynomials as the integration nodes. The error of the formula is given by:

$$\int_{-1}^{1} f(x)dx - \sum_{k=1}^{n} c_k f(x_k) = \frac{2^{2n+1}(n!)^4}{(2n+1)[(2n)!]^3} \, f^{(2n)}(\xi), \tag{3}$$

for some $\xi \in (-1, 1)$, provided f has a continuous $(2n)$th derivative.

The Darboux (or any Riemann) sums method is the most general but the least efficient technique. The number of subintervals required for a given accuracy is usually huge compared with the other formulae described here. On the other

[1] The degree of accuracy of integration formula $\mathrm{Q}f$, is the greatest integer d such that $\mathrm{Q}f$ integrates exactly all polynomials of degree less than or equal to d.

hand, if the Riemann integral exists, this method can always be applied whereas the other methods may fail. Gauss formulae are commonly used because of their accuracy, despite the fact that it may be difficult to obtain an upper bound for the $(2n)$th derivative of f.

3 Numerical Methods for Solving the Initial Value Problem of Ordinary Differential Equations

The initial value problem of the ordinary differential equation may be written in the form:

$$\boldsymbol{y}' = \boldsymbol{f}(t, \boldsymbol{y}), \qquad t \geq a, \qquad \boldsymbol{y}(a) = \boldsymbol{y}_0, \tag{4}$$

where $\boldsymbol{f} : [a, \infty) \times \mathbb{R}^d \to \mathbb{R}^d$, and $\boldsymbol{y}_0$ is an initial condition. The usual requirement is that $\boldsymbol{f}$ satisfies a Lipschitz condition[2], which is sufficient for the initial value problem (4) to have a unique solution $\boldsymbol{y}(t)$. Usually, the problem is to compute a numerical solution of (4) at a point $b > a$. Numerical methods fit such a problem well, because using them we actually do not obtain a continuous approximation of $\boldsymbol{y}(t)$, but calculate the solution $\boldsymbol{y}_0, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_n$ at points $a = t_0 < t_1 < \ldots < t_n = b$, called mesh points, of the compact interval $[a, b]$. In the following we outline Euler's and Runge–Kutta methods, where the mesh points make an equidistant partition of $[a, b]$ $(h = t_{j+1} - t_j = \frac{b-a}{n},\ j = 0 \ldots (n-1))$.

Euler's method Using a Taylor's expansion of $\boldsymbol{y}$ about $t_{j+1},\ j = 0 \ldots (n-1)$ we can write $\boldsymbol{y}(t_{j+1}) \approx \boldsymbol{y}(t_j) + (t_{j+1} - t_j)\boldsymbol{y}'(t_j)$, which gives us:

$$\boldsymbol{y}_{j+1} = \boldsymbol{y}_j + h\boldsymbol{f}(t_j, \boldsymbol{y}_j), \qquad j = 0, \ldots, (n-1), \tag{5}$$

where h is the step size. The error of this method is given by

$$\|\boldsymbol{e}_j^h\| = \|\boldsymbol{y}_j - \boldsymbol{y}(t_j)\| \leq \frac{Bh}{\lambda}(e^{(b-a)\lambda} - 1), \tag{6}$$

where λ is a Lipschitz constant in a given norm $\|\cdot\|$, and B is an upper bound of $\|\boldsymbol{y}''\|$ over the interval $[a, b]$ (since all norms are equivalent in finite dimensional spaces, the choice of the vector norm is not relevant).

Runge–Kutta methods Integrating the differential equation given in (4) over $[t_j, t_{j+1}]$, we obtain $\boldsymbol{y}(t_{j+1}) = \boldsymbol{y}(t_j) + \int_{t_j}^{t_{j+1}} \boldsymbol{f}\,dt$. Using various numerical integration methods to calculate the last integral, we obtain the Runge–Kutta methods. For example, using the trapezoidal rule we obtain the second–order Runge–Kutta method (RK2), while Simpson's rule is used in the construction of the third and the fourth–order Runge–Kutta methods (RK3 and RK4 respectively). These methods are summarized in Table 1.

All of these methods are convergent, i.e. $\|e_j^h\| \to 0$ when $h \to 0$ [14].

[2] A function $\boldsymbol{f} : [a, \infty) \times \mathbb{R}^d \to \mathbb{R}^d$ is said to satisfy a Lipschitz condition in the variable $\boldsymbol{y}$, if there exists a constant $\lambda > 0$ such that

$$\|\boldsymbol{f}(t, \boldsymbol{x}) - \boldsymbol{f}(t, \boldsymbol{y})\| \leq \lambda \|\boldsymbol{x} - \boldsymbol{y}\|,$$

whenever $(t, \boldsymbol{x}), (t, \boldsymbol{y}) \in [a, \infty) \times \mathbb{R}^d$, where $\|\cdot\|$ is a given vector norm.

Table 1. Runge–Kutta methods

	RK2	RK3	RK4
k_1	$h\boldsymbol{f}(t_j, \boldsymbol{y}_j)$	$h\boldsymbol{f}(t_j, \boldsymbol{y}_j)$	$h\boldsymbol{f}(t_j, \boldsymbol{y}_j)$
k_2	$h\boldsymbol{f}(t_{j+1}, \boldsymbol{y}_j + k_1)$	$h\boldsymbol{f}\left(t_j + \frac{1}{2}h, \boldsymbol{y}_j + \frac{1}{2}k_1\right)$	$h\boldsymbol{f}\left(t_j + \frac{1}{2}h, \boldsymbol{y}_j + \frac{1}{2}k_1\right)$
k_3		$h\boldsymbol{f}\left(t_{j+1}, \boldsymbol{y}_j - k_1 + 2k_2\right)$	$h\boldsymbol{f}\left(t_j + \frac{1}{2}h, \boldsymbol{y}_j + \frac{1}{2}k_2\right)$
k_4			$h\boldsymbol{f}(t_{j+1}, \boldsymbol{y}_j + k_3)$
$\boldsymbol{y}_{j+1}$	$\boldsymbol{y}_j + \dfrac{1}{2}[k_1 + k_2]$	$\boldsymbol{y}_j + \dfrac{1}{6}[k_1 + 4k_2 + k_3]$	$\boldsymbol{y}_j + \dfrac{1}{6}[k_1 + 2k_2 + 2k_3 + k_4]$

4 Exact Real Arithmetic Using LFTs

By an exact real arithmetic framework we mean any set of algorithms for computing elementary function up to any required accuracy. One example is Boehm and Cartwright's framework [1, 2], developed and implemented by Ménissier–Morain, [17], which uses B–adic numbers. Another example, which uses linear fractional transformations, is based on the work of Gosper, Vuillemin, Kornerup and Nielsen, and has been developed and implemented on the special base interval $[0, \infty]$ by Edalat and Potts, [8, 20]. Recently, Heckmann [13] has considered the base interval $[-1, 1]$. In this section we summarize the representation of real numbers and functions using LFTs, based on the special base interval $[-1, 1]$.

We work in $\mathbb{R}^* = \mathbb{R} \cup \{\infty\}$, the one–point compactification of $\mathbb{R}$ (as in [22]) identified with the unit circle in the plane as in Fig.1.

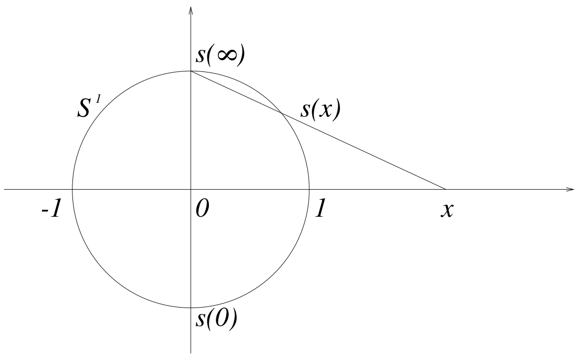

Fig. 1. The stereographic projection

Let us denote the set of vectors, matrices and tensors with integer coefficients respectively by:

$$\mathbb{V} = \left\{ \begin{pmatrix} a \\ b \end{pmatrix} \mid a, b \in \mathbb{Z} \right\}, \qquad \mathbb{M} = \left\{ \begin{pmatrix} a & c \\ b & d \end{pmatrix} \mid a, b, c, d \in \mathbb{Z} \right\},$$
$$\mathbb{T} = \left\{ \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} \mid a, b, c, d, e, f, g, h \in \mathbb{Z} \right\}.$$

Vectors, matrices and tensors induce respectively 0–dimensional LFTs (a fraction in $\mathbb{R}^*$), 1–dimensional LFTs (a function from $\mathbb{R}^*$ to $\mathbb{R}^*$) and 2–dimensional LFTs (a function from $\mathbb{R}^* \times \mathbb{R}^*$ to $\mathbb{R}^*$), which are given by:

$$\Theta \begin{pmatrix} a \\ b \end{pmatrix} = \frac{a}{b}, \qquad \Theta \begin{pmatrix} a & c \\ b & d \end{pmatrix} (x) = \frac{ax + c}{bx + d},$$
$$\Theta \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} (x, y) = \frac{axy + cx + ey + g}{bxy + dx + fy + h}.$$

We can identify an LFT $\Theta(K)$ with K, where K is vector, matrix or tensor. This identification is unique up to scaling: $\Theta(K) = \Theta(kK)$ for any non–zero integer k. The composition of two 1–dimensional LFTs corresponds to matrix multiplication. A non–singular matrix M maps an interval to an interval: the interval $[p, q]$ is mapped to $[Mp, Mq]$ for $\det M > 0$ and $[Mq, Mp]$ for $\det M < 0$. One can verify that for any two intervals with rational end–points, there exists an LFT M such that $M[p, q] = [r, s]$. We say that a matrix M is bounded, respectively refining, on base interval $[-1, 1]$ if $M[-1, 1] \subseteq \mathbb{R}$, respectively $M[-1, 1] \subseteq [-1, 1]$. Similarly, we say that a tensor T is bounded, respectively refining, if $T([-1, 1], [-1, 1]) \subseteq \mathbb{R}$, respectively $T([-1, 1], [-1, 1]) \subseteq [-1, 1]$. The sets of all refining vectors, matrices and tensors with integer coefficients are denoted by $\mathbb{V}^+$, $\mathbb{M}^+$ and $\mathbb{T}^+$ respectively. It is easy to check that for $M, N \in \mathbb{M}$, we have $MN[-1, 1] \subseteq M[-1, 1] \Leftrightarrow N \in \mathbb{M}^+$. A signed normal product (snp), an unsigned normal product (unp), a signed expression tree (sext) and an unsigned expression tree (uext) are respectively defined by the following recursive relations:

$$snp := V \mid M(unp), \qquad sext := V \mid M(uext) \mid T(uext, uext),$$
$$unp := V^+ \mid M^+(unp), \qquad uext := V^+ \mid M^+(uext) \mid T^+(uext, uext),$$

where $V \in \mathbb{V}$, $M \in \mathbb{M}$, $T \in \mathbb{T}$, $V^+ \in \mathbb{V}^+$, $M^+ \in \mathbb{M}^+$ and $T^+ \in \mathbb{T}^+$.

A real number, x, is represented as a shrinking sequence of nested closed intervals with rational end–points, $\{x\} = \bigcap_n [p_n, q_n]$, or using the facts above, as a signed normal product, $\{x\} = \bigcap_n M_n[-1, 1]$. By analogy with the usual representation of real numbers, the first matrix of the normal product, $M_0 \in \mathbb{M}$, is called the sign matrix, while the matrices $M_n \in \mathbb{M}^+$, $n > 0$, are called digit matrices. Edalat and Potts in [6, 8] proposed a standard form, called exact floating form, where the first matrix is one of the four sign matrices S_∞, S_-, S_0, S_+:

$$S_\infty = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \; S_- = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \; S_0 = \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix}, \; S_+ = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

The matrices $M_n,\ n>0$ belong to the finite set of digit matrices in base $r>0$: $A_k^r = \begin{pmatrix} 1 & k \\ 0 & r \end{pmatrix},\ |k|<r$.

One can perform some simple functions, such as $x \mapsto x-1,\ x \mapsto \frac{1}{2x}$, over real numbers using 1–dimensional LFTs. The computation is done by a sequence of emissions and absorptions. A transformation $M \mapsto N_1(N_1^* M)$ is called an emission of a matrix N_1 from a matrix M. Here N_1^* denotes an integer inverse of N_1 (recall that N_1^* is unique up to scaling). In the exact floating form, the first emitted matrix of any expression is one of sign matrices $S_\sigma,\ \sigma \in \{\infty,-,0,+\}$, while the subsequent emitted matrices are digits matrices. On the other hand, by absorption we mean a transformation $MN_2 \mapsto (MN_2)$. Here, matrix M consumes a matrix N_2. Let $S_\sigma D_1 D_2 \ldots$ be an exact floating point representation of a real number x. In order to compute the exact floating expression of $M(x) = \frac{ax+c}{bx+d}$ we first emit a sign matrix $S_{\sigma'}$ and then digit matrices A_k^r from M until the required accuracy is attained. Whenever no emission is possible one has to absorb further digits from x.

While matrices may be used in representation of real numbers by normal product, tensors are used to represent functions by an expression tree. Basic arithmetic operations over two arguments are easily represented by suitable tensors. For example, multiplication xy is represented by $\frac{1xy+0x+0y+0}{0xy+0x+0y+1}$. Furthermore, Edalat and Potts in [8, 20] gave an expression tree representation of a number of elementary functions like sine, cosine, tangent, exponential, logarithm. Their expression tree representation is given in the form as shown in Fig.2. Tensors may emit a matrix in the same way as matrices. But tensors may absorb digits from both of their arguments. These absorptions are called left and right absorption. For more details see [20], where Potts studies various emission and absorption strategies. We say that a tensor T is made of two matrices $T=(T_0,T_1)$, and a matrix M is made of two vectors $M=(M_0,M_1)$. T^T denotes the transpose of a tensor T, obtained by swapping the two middle columns of T. Then, right absorption, $\bullet_R$, and left absorption, $\bullet_L$ are given by:

$$(T_0,T_1) \bullet_R M = (T_0 M, T_1 M), \qquad T \bullet_L M = (T^T \bullet_R M)^T$$

We define, as in [13], the contractivity and the expansivity of a bounded matrix M by $\mathrm{con}M = \sup_{x\in[-1,1]} |M'(x)|$, $\exp M = \inf_{x\in[-1,1]} |M'(x)|$ respectively. For a bounded tensor T, the right contractivity is defined as $\mathrm{con}_R T = \sup_{x\in[-1,1]} \mathrm{con}(T|_x)$, where for fixed x, the 1–dimensional LFT $(T|_x)$ is given by $(T|_x) := (y \mapsto T(x,y))$. Similarly, the left contractivity of T is defined by $\mathrm{con}_L T = \sup_{y\in[-1,1]} \mathrm{con}(T|^y)$. If $\widetilde{T}$ is a result of emission of a digit matrix in base r from a bounded tensor T, then $\mathrm{con}_R\widetilde{T} = r\ \mathrm{con}_R T$ and $\mathrm{con}_L\widetilde{T} = r\ \mathrm{con}_L T$. If, on the other hand, the tensor $\widetilde{T}$ is the result of an absorption of a digit matrix from the right, we have $\mathrm{con}_R\widetilde{T} \le \frac{1}{r}\ \mathrm{con}_R T$ and $\mathrm{con}_L\widetilde{T} \le \mathrm{con}_L T$. Similarly, in the case of absorption from left we have $\mathrm{con}_R\widetilde{T} \le \mathrm{con}_R T$ and $\mathrm{con}_L\widetilde{T} \le \frac{1}{r}\ \mathrm{con}_L T$.

Heckmann in [13] studies the problem of how many input digits in exact floating point form are needed from an argument to produce the required number

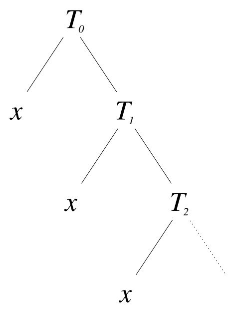

Fig. 2. $f(x)$ represented by an expression tree

of output digits (input complexity). Here we outline some relevant results that we need in order to study the complexity of numerical integration later in this paper. In order to emit k digits in base r of $M(x)$, where M is a refining matrix applied to an argument $x \in [-1,1]$, at least $k + \lceil \log_r(\exp M) \rceil$ and at most $k + \lceil \log_r(2\mathrm{con} M) \rceil$ digits from x have to be absorbed. If the tensor T takes two arguments $x, y \in [-1,1]$, and produces the result in the same interval (refining tensors belong to this class), then in order to produce k digits of $T(x,y)$ in base r, at most $k + \lceil \log_r(4\mathrm{con}_L T) \rceil$ and $k + \lceil \log_r(4\mathrm{con}_R T) \rceil$ digits are needed from x and y respectively.

5 Numerical Integration in Exact Framework

Given an exact real arithmetic framework, to calculate a numerical value of an integral $\mathrm{I}f$ with error not greater than $\epsilon > 0$, we first split ϵ into two parts: $\epsilon = \epsilon_\mathrm{m} + \epsilon_\mathrm{r}$. The number of the integration nodes is obtained by the error ϵ_m of the numerical integration method. Then we compute the approximation $\mathrm{Q}_n f$ which is the value of an elementary function up to ϵ_r accuracy in the exact arithmetic framework.

For example, the number of integration nodes of the compound trapezoidal rule (1) and the compound Simpson's rule (2) are respectively obtained as the integer n such that

$$\frac{B(b-a)^3}{12\epsilon_\mathrm{m}} < n^2 \qquad \text{and} \qquad \frac{C(b-a)^5}{2880\epsilon_\mathrm{m}} < n^4. \tag{7}$$

We then calculate $\mathrm{Q}_1^n f$ by the trapezoidal rule and Q_2^n by Simpson's rule up to precision ϵ_r. Here B and C represent upper bounds of $|f^{(2)}|$ and $|f^{(4)}|$ over $[a,b]$ respectively, and do not have to be calculated exactly.

The computation of the lower Darboux sum for a monotone function f is straightforward; for $\epsilon_\mathrm{m} > 0$ we find an integer n such that $\frac{b-a}{\epsilon_\mathrm{m}} \, |f(b) - f(a)| < n$, and calculate the Darboux sum up to precision ϵ_r. This can be easily extended to a piecewise monotone function if the local minima and maxima of the function are computable. We split the integration region into, say, k subintervals on each of which f is monotonic. We then calculate the integral on each of the subintervals and add them up. The error ϵ_{m} is split into $\epsilon_1, \ldots, \epsilon_k$, and each of these errors is used in order to calculate one of the k integrals. The points where f reaches its k peaks may be obtained by calculating the k roots of f' using the trisection algorithm described by Edalat and Rico in [9].

Comparing the error term of the Gauss–Legendre formula (3) with ϵ_{m}, we can obtain the number of integration nodes required for such accuracy. Therefore, if $f^{(2n)}$ vanishes for some n, the Gauss formulae with n integration nodes will integrate the function f with no error. For example, any polynomial of degree m will be integrated exactly using the Gauss formula with $n \geq \frac{m+1}{2}$. In this case our only concern is the error of the representation ϵ_{r}. We have tabulated the integration nodes (the zeros of Legendre polynomials of degree n) and weights of some Gauss–Legendre integration formulae in the binary sign form in Table 2.

Table 2. The integration nodes and weights of Gauss–Legendre formulae in the binary sign form

n	Integration Nodes	Integration Weights
2	$\bar{1}\bar{1}11\bar{1}11\bar{1}\bar{1}\bar{1}\bar{1}11\bar{1}\bar{1}1\bar{1}11\bar{1}\ldots$	$11111111111111111111\ldots$
	$11\bar{1}\bar{1}1\bar{1}\bar{1}1111\bar{1}\bar{1}11\bar{1}1\bar{1}\bar{1}1\ldots$	$11111111111111111111\ldots$
3	$\bar{1}\bar{1}\bar{1}111\bar{1}\bar{1}11\bar{1}11\bar{1}1\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\ldots$	$10010\bar{1}100100\bar{1}00100\bar{1}0\ldots$
	$\bar{1}1111111111111111111\ldots$	$11100100\bar{1}00100\bar{1}0010\bar{1}\ldots$
	$111\bar{1}\bar{1}\bar{1}11\bar{1}\bar{1}1\bar{1}\bar{1}1\bar{1}11111\ldots$	$10010\bar{1}100100\bar{1}00100\bar{1}0\ldots$
4	$\bar{1}\bar{1}\bar{1}1\bar{1}\bar{1}\bar{1}111\bar{1}\bar{1}\bar{1}11\bar{1}\bar{1}1\bar{1}\bar{1}\ldots$	$1\bar{1}10\bar{1}01\bar{1}00010\bar{1}1\bar{1}0001\ldots$
	$\bar{1}1\bar{1}1\bar{1}1\bar{1}\bar{1}\bar{1}1111\bar{1}111\bar{1}\bar{1}\bar{1}\ldots$	$11\bar{1}010\bar{1}1000\bar{1}010\bar{1}0000\ldots$
	$1\bar{1}1\bar{1}1\bar{1}111\bar{1}\bar{1}\bar{1}\bar{1}1\bar{1}\bar{1}\bar{1}111\ldots$	$11\bar{1}010\bar{1}1000\bar{1}01\bar{1}10000\ldots$
	$111\bar{1}111\bar{1}\bar{1}\bar{1}111\bar{1}\bar{1}11\bar{1}11\ldots$	$1\bar{1}10\bar{1}01\bar{1}00010\bar{1}1\bar{1}0000\ldots$

6 Numerical Integration Using LFTs

Using the LFT framework, we have implemented numerical integration methods and numerical methods for solving the initial value problem of ODEs in C and in the functional programming language Haskell. Here we describe our implementation of numerical integration and give some examples.

All the integration methods described in this work use linear combinations of values of f. Thus for ϵ_{m}, we determine the number of subintervals n and construct an expression tree which represents the nth integration formula $\mathrm{Q}_n f$. Computing the expression tree up to precision ϵ_{r} we obtain the value of the integral $\mathrm{I}f$ within $\epsilon = \epsilon_{\mathrm{m}} + \epsilon_{\mathrm{r}}$ accuracy. The expression tree for $\mathrm{Q}_n f$ is shown in Fig.3, where the number p is given by $p = \lceil \log_2(n+1) \rceil$. Here T_+ is the addition tensor, i.e. $T_+(x, y) = x + y$, while the values $f(x_0), \ldots, f(x_n)$ are as in Fig.2. For the Darboux sums method and the closed Newton–Cotes formulae, the integration coefficients $c_0, \ldots, c_n$ are rational numbers and are represented by a single matrix. In the case of the compound trapezoidal rule, for example, we have $c_0 = c_n = A$, $c_1 = \ldots = c_{n-1} = B$, where A and B are given by:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & n \end{pmatrix}, \qquad B = \begin{pmatrix} 2 & 0 \\ 0 & n \end{pmatrix}.$$

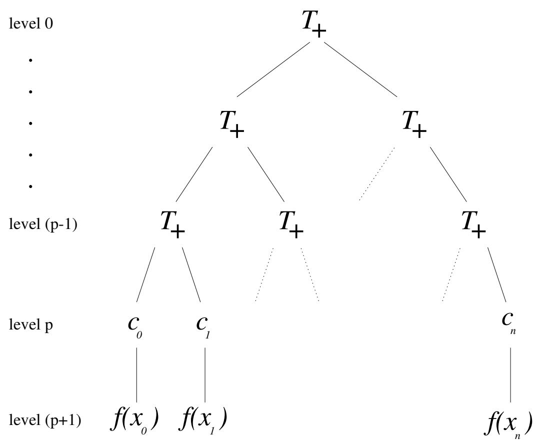

Fig. 3. The expression tree representation of $\mathrm{Q}_n f$

6.1 Case Study: $\int_{-1}^{1} \exp x dx$

In this section we determine how many LFT operations (emissions and absorptions) we need to calculate the value of $\int_{-1}^{1} \exp x dx$ within the given precision $\epsilon = \epsilon_{\mathrm{m}} + \epsilon_{\mathrm{r}}$.

Potts in [20] has given expression tree representation of all basic transcendental functions including the exponential function, which can be represented, for $x \in [-1, 1]$, by the following infinite product:

$$\exp x = S_\infty \prod_{j=0}^{\infty} T_j|_x \quad \text{where} \quad T_j = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 4j+2 \end{pmatrix}.$$

We also use results in [13] in order to determine how many input digits are required to produce, say, k output digits. For example, to produce k output digits in base 2 of $\exp x$ we need in total $\frac{3}{4}k^2 + \frac{13}{2}k + 9$ LFT operations. Provided that

$x+y \in [-1,1]$ T_+ needs $k+2$ digits from each argument in order to produce k output digits in base 2, therefore giving $3k+4$ LFT operations in total. Without loss of generality, we may assume that the result of any addition in Fig.3 is in $[-1,1]$. (Since $\int_{-1}^{1} \exp x dx < 6$, we can multiply the integration weights by $\frac{1}{6}$. Of course, the final result has to be multiplied by 6.) For $n > 3$, in order to produce k output digits in base 2, matrices A and B do not require more than k input digits, giving in total $2k$ LFT operations.

Since digit matrices in base r are refining with contractivity $1/r$, in order to calculate $\mathrm{I}f$ with precision $\epsilon = \epsilon_{\mathrm{m}} + \epsilon_{\mathrm{r}}$, we obtain the number of subintervals n, construct the expression tree as shown in Fig.3 and emit $k = \lceil 1 - \log_2 \epsilon_{\mathrm{r}} \rceil$ digits in base 2. Table 3 gives the upper bound of LFT operations necessary at each level.

Table 3. Number of LFT operations

level	LFT	digits to emit	total LFT operations	copies
0	T_+	k	$3k+4$	1
1	T_+	$k+2$	$3k+10$	2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(p-1)$	T_+	$k+2(p-1)$	$3k+6(p-1)+4$	$\leq 2^{(p-1)}$
p	c_j	$k+2p$	$2k+4p$	$n+1$
$(p+1)$	$f(x_j)$	$k+2p$	$\frac{3}{4}(k+2p)^2 + \frac{13}{2}(k+2p)+9$	$n+1$

We then use these results to determine the upper bound for the total number of LFT operations required to calculate $\mathrm{Q}_n f$. For the Darboux sums method, the trapezoidal rule and Simpson's rule, these are respectively of order:

$$\begin{gathered}
\mathcal{O}\left(\tfrac{1}{\epsilon_{\mathrm{m}}} \cdot \left(\log_2^2 \tfrac{1}{\epsilon_{\mathrm{m}}} + log_2^2 \tfrac{1}{\epsilon_{\mathrm{r}}}\right)\right) \\
\mathcal{O}\left(\sqrt{\tfrac{1}{\epsilon_{\mathrm{m}}}} \cdot \left(\log_2^2 \tfrac{1}{\epsilon_{\mathrm{m}}} + log_2^2 \tfrac{1}{\epsilon_{\mathrm{r}}}\right)\right) \\
\mathcal{O}\left(\sqrt[4]{\tfrac{1}{\epsilon_{\mathrm{m}}}} \cdot \left(\log_2^2 \tfrac{1}{\epsilon_{\mathrm{m}}} + log_2^2 \tfrac{1}{\epsilon_{\mathrm{r}}}\right)\right).
\end{gathered}$$

6.2 Examples

Example 1. In the first two examples, the required precision is divided into two equal parts, $\epsilon_{\mathrm{m}} = \epsilon_{\mathrm{r}} = \frac{1}{2}\,\epsilon$. As expected, Simpson's rule requires the least number of subintervals. Note that the Gauss formula is very accurate with significantly less integration nodes. First we calculate $\int_{-1}^{1} \exp x dx$ with the precision $\epsilon = 2^{-14}$ $(\epsilon_{\mathrm{m}} = \epsilon_{\mathrm{r}} = 2^{-15})$:

method	n	number of nodes	value
Darboux sums	154036	154036	N/A
trapezoidal rule	244	245	$0.235041e1$
Simpson's rule	6	13	$0.235041e1$
Gauss formula	5	5	$0.2350402386e1$
correct value			$0.2350402387e1$

Example 2. For $\epsilon = 2^{-10}$ the approximation values of $\int_{-\frac{1}{2}}^{1} \tan x dx$ are given in the table below:

method	n	number of nodes	value
Darboux sums	6463	6463	N/A
trapezoidal rule	79	80	$0.4851e0$
Simpson's rule	17	35	$0.4850e0$
Gauss formula	5	5	$0.48503e0$
correct value			$0.48504e0$

Example 3. By increasing $\epsilon_{\rm m}$ and hence decreasing $\epsilon_{\rm r}$, the number of subintervals required may fall. But, then we have to emit more digits from the expression. Vice versa, with decreasing $\epsilon_{\rm m}$ and increasing $\epsilon_{\rm r}$. Using Simpson's rule, we integrate $f(x) = \exp x$, over the interval $[-1, 1]$, with constant precision $\epsilon = 2^{-20}$, but with different values $\epsilon_{\rm m}$ and $\epsilon_{\rm r}$.

$\epsilon_{\rm m}$	$\epsilon_{\rm r}$	n	number of nodes	digits required
$63 \cdot 2^{-26}$	2^{-26}	14	29	33
$31 \cdot 2^{-25}$	2^{-25}	14	29	32
$15 \cdot 2^{-24}$	2^{-24}	14	29	31
$7 \cdot 2^{-23}$	2^{-23}	14	29	30
$3 \cdot 2^{-22}$	2^{-22}	15	31	29
2^{-21}	2^{-21}	16	33	28
2^{-22}	$3 \cdot 2^{-22}$	19	39	28
2^{-23}	$7 \cdot 2^{-23}$	23	47	28
2^{-24}	$15 \cdot 2^{-24}$	27	55	28
2^{-25}	$31 \cdot 2^{-25}$	32	65	28
2^{-26}	$63 \cdot 2^{-26}$	38	77	27

7 Initial Value Problem in Exact Framework

In non–exact frameworks, round–off errors play a significant role in the choice of the step size h. As h becomes smaller, more computation is needed and round–off errors may have more influence on the final result. Furthermore, for sufficiently small h, the error will not decrease, but will become larger, i.e. the method will not converge to the exact solution. Fortunately, in exact frameworks we do not have such a problem. We can solve the initial value problem of ODEs in an exact framework following the same idea as in the numerical integration methods described in the previous section. To obtain the numerical value of the solution $\boldsymbol{y}(b)$ with a given accuracy $\epsilon = \epsilon_{\mathrm{m}} + \epsilon_{\mathrm{r}}$, we use the error term of Euler's method given by relation (6). Provided that we have a Lipschitz constant λ and an upper bound B for $\|\boldsymbol{y}''\|$, we calculate an integer n such that:

$$n \geq \frac{B(b-a)(e^{(b-a)\lambda} - 1)}{\lambda \epsilon_{\mathrm{m}}}.$$

The step size, given by $h = (b-a)/n$, will guarantee that the global error is smaller than ϵ_{m}. We calculate the value $\boldsymbol{y}_n$, which is an approximation of $\boldsymbol{y}(b)$, up to precision ϵ_{r}. Unfortunately, very often this error is too conservative to be exploited. For example, to calculate $y(1)$ with accuracy $\epsilon_{\mathrm{m}} = 10^{-3}$, where y is the solution of the initial value problem $y' = -100y,\ y(0) = 1$, we have $h < 10^{-48}$.

7.1 Examples

Example 4. In this example we use Euler's method and Runge–Kutta methods (RK2, RK3, RK4), implemented in exact framework using LFTs, to obtain $y(1)$, where y is the solution of the initial value problem given by:

$$y' = y^2\left(\frac{1}{x^2} - 1\right), \quad y\left(\frac{1}{2}\right) = \frac{2}{7}.$$

The step size is $h = \frac{1}{20}$, and the exact solution is given by $y = \frac{x}{x^2+x+1}$.

x	Euler's	RK2	RK3	RK4	exact value
0.50	0.2857143	0.2857143	0.2857143	0.2857143	0.2857143
0.55	0.2979592	0.2969544	0.2968981	0.2968962	0.2968961
0.60	0.3081945	0.3062298	0.3061259	0.3061226	0.3061225
0.65	0.3166375	0.3137789	0.3136355	0.3136310	0.3136309
0.70	0.3234896	0.3198163	0.3196401	0.3196349	0.3196347
0.75	0.3289354	0.3245333	0.3243303	0.3243245	0.3243243
0.80	0.3331431	0.3281000	0.3278753	0.3278691	0.3278688
0.85	0.3362646	0.3306666	0.3304247	0.3304181	0.3304179
0.90	0.3384361	0.3323659	0.3321103	0.3321035	0.3321033
0.95	0.3397794	0.3333144	0.3330484	0.3330414	0.3330412
1.00	0.3404031	0.3336144	0.3333406	0.3333336	0.3333333

Example 5. We use RK4 over the interval $[0, 2]$ with step size $h = \frac{1}{5}$ to obtain approximations of the initial value problem:

$$y' = -y + x^2 + 2x, \quad y(0) = 1.$$

The exact solution of the problem is $y = e^{-x} + x^2$.

x	RK4	exact value	error
0.0	1.000000000000000000	1.000000000000000000	0.000000000000000000
0.2	0.858739999999999999	0.858730753077981858	0.000009246922018141
0.4	0.830336395999999999	0.830320046035639300	0.000016349964360699
0.6	0.908833418618399999	0.908811636094026432	0.000021782524373567
0.8	1.089354880936838026	1.089328964117221591	0.000025916819616435
1.0	1.367908486185687187	1.367879441171442321	0.000029045014244865
1.2	1.741225607923094956	1.741194211912202096	0.000031396010892859
1.4	2.206630112726901943	2.206596963941606476	0.000033148785295466
1.6	2.761930960959938851	2.761896517994655408	0.000034442965283443
1.8	3.405334275436600602	3.405298888221586538	0.000035387215014064
2.0	4.135371349109126133	4.135335283236612691	0.000036065872513441

Acknowledgements

We would like to thank Lindsay Errington for his C implementation of exact reals using LFTs. The second author would like to thank Marko Vrdoljak for discussions.

References

1. Boehm, H. J., Cartwright, R.: Exact Real Arithmetic: Formulating Real Numbers as Functions. In Turner, D., editor, Research Topics in Functional Programming, Addison–Wesley (1990) 43–64.
2. Boehm, H. J., Cartwright, R., Riggle, M., O'Donnell, M. J.: Exact Real Arithmetic: A Case Study in Higher Order Programming. ACM Symposium on Lisp and Functional Programming (1986).
3. Collatz, L.: The Numerical Treatment of Differential Equations. Springer-Verlag, Berlin Heidelberg New York Tokyo (1966).
4. Davis, P. J., Rabinowitz, P.: Methods of Numerical Integration. Academic Press, London New York (1975).
5. Di Gianantonio, P.: A functional approach to real number computation. PhD Thesis, University of Pisa (1993).
6. Edalat, A.: Domains for Computation in Mathemetics, Physics and Exact Real Arithmetic. Bulletin of Symbolic Logic, Vol. 3 (1997).
7. Edalat, A., Escardó, M. H.: Integration in Real PCF. Eleventh Annual IEEE Symposium on Logic in Computer Science (LICS) (1996).
8. Edalat, A., Potts, P. J.: A New Representation for Exact Real Numbers. Electronic Notes in Theoretical Computer Science, Proceedings of Mathematical Foundations of Programming Semantics 13 (1997).
9. Edalat, A., Rico, F.: Two Algorithms for Root Finding in Exact Real Arithmetic. Third Real Numbers and Computers Conference (1998) 27-44.
10. Engels, H.: Numerical Quadrature and Cubature. Academic Press, London New York (1980).
11. Escardó, M. H.: PCF extended with real numbers. Theoretical Computer Science, 162(1):79–115 (1996).

12. Gosper, W.: Continued Fraction Arithmetic. HAKMEM Item 101B, MIT Artificial Intelligence Memo 239, MIT (1972).
13. Heckmann, R.: How Many Argument Digits are Needed to Produce n Result Digits? To appear ENTCS (1999).
14. Iserles, A.: A First Course in the Numerical Analysis of Differential Equations. Cambridge Texts in Applied Mathematics, Cambridge University Press (1996).
15. Krommer, A. R., Ueberhuber, C. W.: Numerical Integration on Advanced Computer Systems. LNCS, Vol. 848, Springer-Verlag, Berlin Heidelberg New York Tokyo (1994).
16. Krylov, V. I.: Approximate Calculation of Integrals. Macmillan, New York London (1962).
17. Ménissier–Morain, V.: Arbitrary Precision Real Arithmetic: Design and Algorithms. Submitted to the Journal of Symbolic Computation (1996).
18. Nakamura, S.: Applied Numerical Methods with Software. Prentice Hall, Englewood Cliffs, New Jersey (1991).
19. Nielsen, A., Kornerup P.: MSB–First Digit Serial Arithmetic. Journal of Univ. Comp. Science, 1(7):523–543 (1995).
20. Potts, P. J.: Exact Real Arithmetic Using Möbius Transformations. PhD Thesis, University of London, Imperial College (1998).
21. Simpson, A. K.: Lazy Functional Algorithms for Exact Real Functionals. Mathematical Foundations of Computer Science, Springer LNCS 1450:456–464 (1998).
22. Vuillemin, J. E.: Exact Real Computer Arithmetic with Continued Fractions. IEEE Transactions on Computers, 39(8):1087–1105 (1990).