, , 1–12 ()

Accelerated Shift-and-Add algorithms

N. REVOL AND J.-C. YAKOUBSOHN nathalie.revol@univ-lille1.fr, yak@cict.fr
Lab. ANO, Univ. de Lille 1, UFR IEEA, Bat. M3, 59655 Villeneuve d'Ascq Cedex, FRANCE
Lab. d'Analyse Numérique et Optimisation, Univ. Paul Sabatier, 118 rte de Narbonne, 31062 Toulouse Cedex 4, FRANCE

Editor:

Abstract. The problem addressed in this paper is the computation of elementary functions (exponential, logarithm, trigonometric functions, hyperbolic functions and their reciprocals) in fixed precision, typically the computer single or double precision.The method proposed here combines Shift-and-Add algorithms and classical methods for the numerical integration of ODEs: it consists in performing the Shift-and-Add iteration until a point close enough to the argument is reached, thus only one step of Euler method or Runge-Kutta method is performed. This speeds up the computation whilst ensuring the desired accuracy is preserved. Time estimations on various processors are presented which illustrate the advantage of this hybrid method.

Keywords: single and double floating-point numbers, computation of elementary functions, Shift-and-Add algorithms, numerical integration of ODEs

1. Introduction

Nowadays, one is so accustomed to have a pocket calculator performing additions and multiplications as well as square roots or exponentials that he does not wonder how these operations are performed.

Arithmetic operations ($+$, $-$, $*$, $/$) are generally implemented according to methods learnt at primary school. The complexity of an arithmetic operation (the number of operations performed on digits) is linear for the addition/subtraction and quadratic for the multiplication and division (even if division is much slower from a practical point of view). Asymptotically faster algorithms have been derived for the latter ones, but they supersede the "school" ones only for operands with much more figures than a pocket calculator can usually cope with, and are therefore not used.

If we now turn to so-called transcendental or elementary functions (exp, log, sin, arctan, sinh, arccosh,...), it is not so obvious to decide how to compute them. There is no way to compute them exactly with a finite number of arithmetic operations and thus approximations are sought. These approximations are computed in different ways, depending on the required accuracy for the result. If the number of figures of the result is not fixed, but depends on the user's needs, then solutions can be to use a Taylor expansion, a Padé approximant or the algorithms developped by Brent in [3].

In this paper we are interested with fixed precision, of the kind provided by the "float" type on any computer. Of course, the previously listed methods can be employed (and for instance Padé approximants, or rather continued fractions, were

used in early pocket calculators). However, since a fixed number of digits is required, pre-computed tables can be used; these tables have a fixed (small) number of entries and these entries use the same fixed storage. Thus specific methods have been developped for the fixed-precision computations of elementary functions. Among such methods, we will be interested only in Shift-and-Add ones; for an excellent introduction to the computation of elementary functions, see [9]. A Shift-and-Add algorithm decomposes its argument into a number basis such that the decomposition is performed by means of additions only. Furthermore, the function to be computed is computed along with the decomposition and requires only additions and multiplications or divisions by 2, which are realized on a computer by shifts. Since additions and shifts are very efficiently performed, the elementary iteration is very efficient. Moreover, the number of iterations is small.

A Shift-and-Add computation looks like integrating the ODE the function satisfies, with fixed, decreasing steps, until the argument is reached. In this paper we propose a method which combines the advantages of Shift-and-Add and numerical integration: it consists in performing the Shift-and-Add iteration until the reached point is close enough to the argument to ensure the convergence of the numerical integration of the ODE satisfied by the computed function; when the distance between these two points is close enough, only one step of a numerical integration method suffices to reach the argument whilst ensuring the desired accuracy is preserved, and the remaining steps of the Shift-and-Add algorithm can be shunted.

It has to be mentioned that any of these methods (for infinite precision as well as for fixed precision) requires that the argument belongs to a fixed interval $[A; B]$. Techniques have been developed to ensure that the argument belongs to it, they are called *argument reduction* and are the subject of many papers [4, 5, 6, 11]. It will be assumed in the following that the argument is already reduced.

This paper is composed as follows: in a first part we recall explicit Euler and Runge-Kutta-4 methods for the integration of an ODE [1] and we precise some useful quantities. In a second part, Shift-and-Add methods will be introduced; this name covers algorithms for exponential and logarithm as well as CORDIC method for trigonometric functions. Then our "hybrid" method will be stated in a general framework. It will then be specified for various elementary functions and corresponding experimental results and estimated times for two different processors will be presented. Finally, redundant Shift-and-Add methods will be introduced and it will be shown that our method is suited to a redundant number system without further work.

2. Numerical integration of ODEs

2.1. Problem

An ODE (of first-order, in explicit form) is an equation

$$y' = f(t, y)$$

where y' is the first derivative of y, y being a function of one (one-dimensional) variable t to $\mathbb{R}^p$ and f is a given function defined on a domain $D_f \subset \mathbb{R} \times \mathbb{R}^p$ to $\mathbb{R}^p$. Usually, an initial condition is specified : $y(t_0) = \eta \in \mathbb{R}^p$.

The corresponding Cauchy problem or initial value problem is the following:

$$\text{solve } \left\{ \begin{array}{lcl} y' & = & f(t,y), \\ y(t_0) & = & \eta. \end{array} \right.$$

Let $[\alpha;\beta] \subset \mathbb{R}$ with $t_0 \in [\alpha;\beta]$, a $\mathcal{C}^1$-function $Y : \ t \in [\alpha,\beta] \mapsto Y(t) \in \mathbb{R}^p$ is a solution if

$$\left\{ \begin{array}{l} (t,Y(t)) \in D_f \ \forall t \in [\alpha;\beta] \\ Y(t_0) = \eta \\ Y'(t) = f(t,Y(t)) \ \forall t \in [\alpha;\beta] \end{array} \right. \quad \text{are fulfilled.}$$

Cauchy-Lipschitz theorem states the existence of a solution if f is a continuous function, and furthermore its uniqueness if f is also local Lipschitz relative to y. These two conditions hold for the elementary functions addressed in this paper.

To integrate an ODE, numerical methods obtain approximate values of the solution at a set of points $t_0 < t_1 < \ldots < t_n < \ldots < t_N$. The approximate value y_n of $Y(t_n)$ is computed by using some of the values obtained in previous steps. In what follows, only explicit Euler and Runge-Kutta of order 4 are considered.

For more definitions and results, see for instance [1], particularly to obtain precise definitions of error and estimations for the following methods.

2.2. Explicit Euler and RK4 methods

Let us denote by $Y(t)$ the solution of the initial value problem. Explicit Euler method is a single-step method of order 1 defined on $[t_0;\gamma]$ by

$$\left\{ \begin{array}{lcl} y_{n+1} & = & y_n + hf(t_n,y_n), \\ y_0 & = & \eta, \end{array} \right.$$

where $h = \frac{\gamma - t_0}{N}$ and for any $0 \leq n \leq N$, $t_n = t_0 + nh$.

A Runge-Kutta method aims at achieving a given order without evaluating any derivatives of f (which can be cumbersome both to establish and to evaluate) and involves the evaluation of f at intermediary points. Among these methods, RK4 is a most popular one since it is of order 4 and requires only 4 evaluations of f. It is given by the following formulae

$$\begin{array}{lcl} k_1 & = & f(t_n,y_n) \\ k_2 & = & f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\ k_3 & = & f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\ k_4 & = & f(t_n + h, y_n + hk_3) \\ y_{n+1} & = & y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \\ \\ y_0 & = & \eta \end{array}$$

2.3. Choice of $\boldsymbol{h}$ *to achieve a given precision*

In this paragraph we consider the computation of $Y(t)$ for any $t \in [t_0;\gamma]$. The method we propose in §3 is based, on a first part, on Shift-and-Add methods which compute exactly (up to rounding errors, which are not taken into account) $Y(t_k)$ where t_k is an intermediary point used to reach t. This explains why we consider only the error $y_n - Y(t_n)$ in order to choose h. For Euler, h has to satisfy $h \leq \sqrt{\frac{2\varepsilon}{\|y''\|_\infty}}$ (the maximum for $\|.\|_\infty$ is taken on $[t_0;\gamma]$) and for RK4 h has to satisfy a more complex inequality: $h \leq \sqrt[5]{\frac{2880\varepsilon}{49\|\frac{\partial^4 f(t,y(t))}{\partial t^4}\|_\infty}}$. The following table sums up the choice of h for a given precision ε in a first column, $\varepsilon = 2^{-24}$ corresponding then to the single precision in a second column and $\varepsilon = 2^{-53}$ for the double precision in a third column, for each elementary function and for explicit Euler and RK4. For calculations details, see [12].

Function on $[t_0;\gamma]$	Euler ε	Euler single	Euler double	RK4 ε	RK4 single	RK4 double
exp $[0;1.56]$	$\sqrt{\frac{2\varepsilon}{e^{1.56}}}$	$1.58\,10^{-4}$	$6.83\,10^{-9}$	$\sqrt[5]{\frac{120\varepsilon}{e^{1.56}+11e^{3.12}}}$	$3.10\,10^{-2}$	$5.55\,10^{-4}$
ln $[1;2[$	$\sqrt{2\varepsilon}$	$3.45\,10^{-4}$	$1.49\,10^{-7}$	$\sqrt[5]{\frac{120\varepsilon}{49}}$	$4.30\,10^{-2}$	$7.71\,10^{-4}$
$\begin{pmatrix}\sin t\\ \cos t\end{pmatrix}$ $[0;\frac{\pi}{4}]$	$\sqrt{2\varepsilon}$	$3.45\,10^{-4}$	$1.49\,10^{-7}$	$\sqrt[5]{\frac{320\varepsilon}{129}}$	$4.31\,10^{-2}$	$7.23\,10^{-4}$
arctan $[0;1]$	$\sqrt{2\varepsilon}$	$3.45\,10^{-4}$	$1.49\,10^{-7}$	$\sqrt[5]{\frac{120\varepsilon}{29}}$	$4.77\,10^{-2}$	$8.56\,10^{-4}$
$\begin{pmatrix}\sinh t\\ \cosh t\end{pmatrix}$ $[0;1]$	$\sqrt{\frac{2\varepsilon}{\cosh 2}}$	$1.78\,10^{-4}$	$7.68\,10^{-8}$	$\frac{1}{\cosh 2}\sqrt[5]{\frac{320\varepsilon}{129}}$	$1.14\,10^{-2}$	$2.05\,10^{-4}$
argtanh $[0;0.76]$	$\sqrt{\frac{2\varepsilon}{8.52}}$	$1.17\,10^{-4}$	$5.08\,10^{-8}$	$\sqrt[5]{\frac{30\varepsilon}{7693}}$	$1.19\,10^{-2}$	$2.13\,10^{-4}$

(Trigonometric and hyperbolic sines and cosines have to be treated in pairs since the derivative of a sine is (up to a sign) a cosine and vice-versa.) What these bounds should put in evidence is the following fact: in order to compute $\exp t$ with a precision 10^{-16} (which is roughly the double-precision accuracy on a computer), even a "good" method such as RK4 requires about 10,000 steps. This shows that numerical integration methods are not good candidates for computing the elemen-

tary functions. However we will show how they can be used in order to "shunt" some other methods' iterations.

3. Shift-and-add algorithms

3.1. Principle of Shift-and-Add algorithms

In this section, we study methods for computing the elementary functions without multiplications and divisions (thus only operations with linear complexity are performed) and with a small number of iterations. These methods are very well suited to hardware computations. Our presentation will follow [8] and [9].
We will firstly illustrate the underlying principle with the computation of $\exp t$, $t \in [0; b]$ (b has to be precised). Since we can write t as a sum $\sum_{k=0}^{+\infty} d_k \ln(1 + 2^{-k})$ with the values $\ln(1 + 2^{-k})$ being tabulated and either $d_k = 0$ or 1, then $\exp t = \prod_{k=0}^{+\infty}(1 + 2^{-k})^{d_k}$. (For the proof of this assertion and the following ones about the decomposition of a number as a sum or product of elements of a discrete basis, see [8].) The computation of $\exp t$ based on this formula involves only shifts and additions, thus the inner statement is very efficiently performed on hardware.

Such a decomposition is a generalization of more familiar ones: on a computer, one is accustomed to write a number $z \in [0; 2]$ as $z = \sum_{k \in I\!N} d_k 2^{-k}$. Shift-and-add algorithms use bases that are well suited to the computation of elementary functions.

3.2. Exponential and logarithm

Using the results presented in the previous paragraph, the algorithm computing $\exp t$ can be derived as follows, the stopping criterion being based on the property $0 \leq t - t_k \leq 2^{-k+1}$:

---

input: $t \in [0; 1.56\ldots]$ *; precision* ε
output: $\exp t$ *approximated (more and more) accurately by* e_k
$t_0 := 0$*;* $e_0 := 1.0$*;* $k := 0$*;*
while $2^{-k+1} > \varepsilon$ *do*
$d_k := \begin{cases} 1 & \text{if } t_k + \ln(1 + 2^{-k}) \leq t \\ 0 & \text{otherwise} \end{cases}$
$t_{k+1} := t_k + d_k \ln(1 + 2^{-k})$
$e_{k+1} := e_k + d_k 2^{-k} e_k$
$k := k + 1$
end loop

An invariant of this algorithm is that $e_k = \exp t_k \ \forall k$.

This method is a step-by-step method, each step aiming at getting closer to the argument. However, it is different from a numerical integration method since the steps are imposed (they are the elements of the discrete basis). Moreover, to compare this with the example at the end of the previous section, to get an accuracy

of $10^{-16} \simeq 2^{-53}$ (which is the double precision accuracy on a computer), only 54 iterations are required instead of tens of thousands!

The same algorithm can be used to compute the logarithm, at the expense of a slight modification: if t becomes the result and e the input, the invariant $e_k = \exp t_k$ can also be written $t_k = \ln e_k$ and thus the logarithm can be computed. The only problem arises when $t_k + \ln(1 + 2^{-k})$ is compared to t since t is now unknown. However, this comparison is equivalent to $e_k(1 + 2^{-k}) \leq \exp t = e$ since exp is an increasing function.

3.3. Trigonometric functions

The underlying idea to compute the sine or cosine of an angle θ is to decompose it as a sum: $\theta = \sum_k \theta_k$. Since the sine of the sum of two angles involves their cosine, sine and cosine will be computed together and thus it corresponds to computing a rotation of angle θ as a succession of rotations of angles θ_k. To simplify the formulae, $(\theta_k)_{k \in I\!N}$ is chosen as $(\arctan 2^{-k})_{k \in I\!N}$.

The decomposition of θ as $\sum d_k \theta_k$ with $d_k \in \{-1; 1\}$ leads to an algorithm which again involves only shifts and additions (and a final multiplication). It is known as CORDIC algorithm (COordinate Rotation Digital Computer) and is due to Volder [14]. It can also be used to compute the arctan function. See for instance [9] for a detailed presentation.

Using the fact that $\arctan 2^{-k} \leq 2^{-k}\ \forall k$, it can be shown that $k + 1$ steps are necessary to obtain k digits of the result (the absolute error is bounded by 2^{-k+1} and again rounding errors are not taken into account).

3.4. Hyperbolic functions

Since trigonometric and hyperbolic formulae are very similar, it seems possible to use the same kind of algorithm, replacing $\arctan 2^{-k}$ by $\operatorname{argtanh} 2^{-k}$, to compute the hyperbolic functions. J. Walther in 1971 showed that the sequence $(\operatorname{argtanh} 2^{-k})_{k \in I\!N}$ does not satisfy the conditions enabling to compute a signed-digit decomposition. He also discovered that if certain terms of the sequence are repeated (namely, the terms $\frac{3^{k+1}-1}{2}$) then the resulting sequence is a discrete basis. With this basis, a slight adaptation of CORDIC algorithm computes the cosh, sinh and argtanh functions.

4. Fast computation of elementary functions

4.1. General idea

In the two previous sections we have recalled what numeric integration and Shift-and-Add methods are. We will now combine them.

The advantage of a Shift-and-Add method is the small number of steps it requires to obtain a given accuracy. Each step makes the current argument closer to the

actual one, and the main drawback is that the steps tend to be smaller and smaller. When the distance between the current argument and the actual one is small enough to ensure the convergence of a numerical integration method with the required accuracy, we will drop off the Shift-and-Add method and go directly to the actual argument thanks to the numerical integration method. This hybrid method still has a drawback: the computations involved by the last step are no more shifts and additions. However, the number of spared Shift-and-Add steps is usually so important that some time can be lost in the computations of the last step.

More precisely, suppose $Y(t)$ has to be computed with a required accuracy 2^{-n}. At each step k of the Shift-and-Add algorithm, the absolute error on t is $|t - t_k| \leq 2^{-k+1}$ with $t_k = \sum_{j=0}^{k} d_k w_k$ and the value computed at step k, y_k, is exactly equal to $Y(t_k)$ (again, rounding errors are not taken into account). It is possible to determine the step h of the numerical integration method corresponding to a method error less than 2^{-n}. Let M be the smallest integer such that $h \geq 2^{-M+1}$. If the Shift-and-Add algorithm is stopped after the M-th iteration, giving t_M and y_M and if one iteration of the numerical integration method is performed with a step $h' = t - t_M$, then the error is bounded by $|y_{M+1} - Y(t)| \leq 2^{-n}$.

We will now precise the value of M corresponding to the single (32 bits) and double (64 bits) IEEE floating-point numbers, for the various elementary functions when the numerical integration method is either explicit Euler or RK4. Error has to be less than 2^{-24} for the single precision type and less than 2^{-53} for the double precision type. For instance, the hybrid algorithms to compute the exponential are given on the following page.

Function	Euler		RK4	
	M_E Single prec.	M_E Double prec.	M_{RK4} Single prec.	M_{RK4} Double prec.
exp	14	29	7	12
ln	13	24	6	12
$\begin{pmatrix} \sin t \\ \cos t \end{pmatrix}$	13	24	6	12
arctan	13	24	6	12
$\begin{pmatrix} \sinh t \\ \cosh t \end{pmatrix}$	14	25	8	14
argtanh	15	26	8	14

Experimental times have been obtained with C codes for the elementary functions. The hardware FPU is much faster than the software Shift-and-Add (with a factor $\simeq 50$). The Euler-hybrid is faster, with a factor between 1.5 and 2, and the RK4-hybrid is faster than the software Shift-and-Add with a factor between 2.5 and 4. (The following times are averaged times measured on a 166MHz Pentium processor, the average is taken over the computed $f(x)$ where f is the elementary function and x takes 15,000 different values).

Function	Method	Single precision time in μs	Double precision time in μs
exp on $[0; 1.56]$	FPU	1	2
	Shift-and-Add	23	33
	hybrid Euler	10	20
	hybrid RK4	6	9
ln on $[1; 2]$	FPU	1	2
	Shift-and-Add	30	61
	hybrid Euler	19	28
	hybrid RK4	10	18
sin cos on $[0; \frac{\pi}{4}]$	FPU	1	2
	Shift-and-Add	57	117
	hybrid Euler	31	55
	hybrid RK4	17	30
arctan on $[0; 1]$	FPU	1	2
	Shift-and-Add	51	110
	hybrid Euler	31	52
	hybrid RK4	17	30
sinh cosh on $[0; 1]$	FPU	2	4
	Shift-and-Add	55	120
	hybrid Euler	31	57
	hybrid RK4	22	34
argtanh on $[0; 0.76]$	FPU	1	2
	Shift-and-Add	57	116
	hybrid Euler	36	62
	hybrid RK4	22	35

Below are the hybrid algorithms to compute the exponential:

Euler-hybrid	RK4-hybrid

$t_0 := 0$; $y_0 := 1$
for $k = 0$ to $M_E - 1$ do
$d_k := \begin{cases} 1 & \text{if } t_k + w_k \leq t \\ 0 & \text{otherwise} \end{cases}$
$t_{k+1} := t_k + d_k w_k$
$y_{k+1} := y_k + d_k 2^{-k} y_k$
end for
$h' := t - t_{M_E}$
$y := y_{M_E} + h' y_{M_E}$
	$t_0 := 0$; $y_0 := 1$
	for $k = 0$ to $M_{RK4} - 1$ do
	$d_k := \begin{cases} 1 & \text{if } t_k + w_k \leq t \\ 0 & \text{otherwise} \end{cases}$
	$t_{k+1} := t_k + d_k w_k$
	$y_{k+1} := y_k + d_k 2^{-k} y_k$
	end for
	$h' := t - t_{M_{RK4}}$
	$y := y_{M_{RK4}} \left[1 + h' \left(1 + \frac{h'}{2}[1 + \frac{h'}{3}(1 + \frac{h'}{4})]\right)\right]$

If we try to estimate the number of clock cycles for each method, on a hypothetical processor such that an addition, a multiplication or a shift are performed in one clock cycle whereas a division is five times longer, we observe that this is in good concordance with the experimental times and also that our hybrid methods supersede the Shift-and-Add ones by a multiplicative factor between 2 and 3.5. We also estimate the number of clock cycles with a less favourable configuration, the PA-8000 processor from HP, PA-RISC 2.0: it performs a shift in one clock cycle, an addition or a multiplication in 3 clock cycles, a single precision division in 17 clock cycles and a double precision division in 31 clock cycles. For this processor, our hybrid algorithms always improve the Shift-and-Add ones. Furthermore, the hybrid RK4 is superior to the hybrid Euler except when the last step involves too many divisions (cf. the logarithm): when a division is very costly, in terms of time, then the hybrid Euler should be preferred. In the following table, times are indicated in clock cycles.

Function	Method	Single precision		Double precision	
		hyp. proc.	PA-8000	hyp. proc.	PA-8000
exp	Shift-and-Add	75	175	162	378
on $[0; 1.56]$	hybrid Euler	45	107	90	212
	hybrid RK4	37	95	55	151
ln	Shift-and-Add	75	175	162	378
on $[1; 2]$	hybrid Euler	42	114	75	205
	hybrid RK4	45	129	63	227
sin	Shift-and-Add	125	275	270	594
cos	hybrid Euler	69	155	124	276
on $[0; \frac{\pi}{4}]$	hybrid RK4	49	121	79	201
arctan	Shift-and-Add	125	275	270	594
on $[0; 1]$	hybrid Euler	78	186	133	335
	hybrid RK4	53	139	83	247
sinh	Shift-and-Add	125	275	270	594
cosh	hybrid Euler	74	166	129	287
on $[0; 1]$	hybrid RK4	43	103	73	183
argtanh	Shift-and-Add	125	275	270	594
on $[0; 0.76]$	hybrid Euler	88	208	138	346
	hybrid RK4	68	178	98	300

5. Redundant number systems

In the previous sections, it has been shown that using a suitable number basis enables to perform quickly various computations. Redundant number systems have been introduced to enable the computation of an addition in constant time. We

will firstly define a (radix 2) redundant number system, sketch its main features and the computation of the elementary functions with such a system. Finally it will be shown how to adapt our hybrid methods to a redundant number system.

5.1. Redundant number systems

Redundant number systems have been introduced to enable fully parallel additions. What prevents the addition from being parallel is the propagation of the carry. To avoid this, Avizienis proposed to represent numbers with *signed digits* -1, 0 or 1. For instance, the number 5 can be written as 101, $1\bar{1}01$ or $10\bar{1}\bar{1}$ in radix 2, where $\bar{1}$ denotes -1. This variety of representations explains the name of this number system. Actually, Avizienis [2] proposed redundant number systems for any radix, but we will focus only on the radix-2 system.

Another kind of redundant system relies on the replacement of every digit by two digits, the second one being a lazy representation of the carry, *i.e.* no carry is propagated. Such a system is called a *carry-save* number system.

Redundant number systems are commonly used by microprocessors in their multipliers or dividers (cf. the Pentium divider). The main advantages of redundant number systems is to enable constant time additions. Since a multiplication consists in a series of additions, it can be performed in linear time. Divisions are also performed in linear time in a redundant number system. However, it is difficult to perform a comparison: since a given number can be written in different ways, comparing the representations of two numbers does not suffice to state their identity or their inequality. It is then necessary to write the arguments of a comparison in a canonical way (which can be their "classical" representations) to be able to compare them. For a more detailed introduction to redundant number systems and the corresponding arithmetic operators, see [2, 7, 9, 10, 13].

5.2. Computing the elementary functions

Let us present the principle of the algorithm for the computation of the exponential. The "classical" Shift-and-Add algorithm is (here, $L_k = t - t_k$)

$$L_0 := t;\ E_0 := 1.0$$
$$d_k := \begin{cases} 1 & \textit{if } L_k \geq \ln(1+2^{-k}) \\ 0 & \textit{otherwise} \end{cases}$$
$$L_{k+1} := L_k - \ln(1 + d_k 2^{-k})$$
$$E_{k+1} := E_k(1 + d_k 2^{-k})$$

In order to adapt this algorithm for a redundant number system, we have to avoid the comparison $L_k \geq \ln(1+2^{-k})$. Indeed, such a comparison can be performed in linear time; since the aim here is to accelerate the Shift-and-Add algorithm, linear time operations must be avoided. We thus allow the d_k to belong to $\{-1; 0; 1\}$ and they have to be chosen so that L_k tends to zero. This signed digit d_k can be determined by performing an approximate comparison: since only four digits of L_k have to be examined, this "comparison" is done in constant time.

The same kind of algorithm exists for every elementary function.

5.3. Fast redundant methods to compute the elementary functions

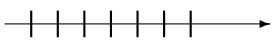
To adapt this algorithm to our hybrid procedure, we only need to know when to jump from this method to a numerical method for integrating an ODE, *i.e.* we need to know a bound for L_k: this bound is given by $-\frac{5}{2}2^{-k} < L_k < 2^{-k+1}$ for the exponential and by similar properties for the other functions [9, 10, 13]. It is therefore straightforward to determine the number of iterations of the Shift-and-Add algorithm to perform: for instance, to compute the exponential in double precision with the hybrid RK4 method, to get $|L_n| \leq 5.55\ 10^{-4}$ only $N = 13$ iterations of the Shift-and-Add algorithm suffice.

We will now sum up in a table the number of steps of the redundant Shift-and-Add algorithm required to compute the exponential and logarithm in single and double precision with our hybrid procedure:

Function	Single precision		Double precision	
	Euler	RK4	Euler	RK4
exp	11	7	29	13
ln	14	7	26	13

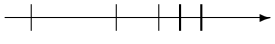
6. Conclusion

In this paper, we have studied how to accelerate the Shift-and-Add algorithms by substituting some iterations by one step of an ODE integration method. This combines the advantages of the two kinds of methods, without suffering their drawbacks: an ODE integration method performs small steps



whereas a Shift-and-Add method performs steps whose length is (roughly) divided by 2 at each iteration

The idea is to jump from this method to an ODE integration method when the step of the Shift-and-Add method becomes too small.



The number of spared steps of the Shift-and-Add algorithm are given in this paper, the formulae for the length of the unique step of explicit Euler or RK4 also: to determine precisely what the gain is on a given processor, one has to compare the number of clock cycles required by each method.

Even if we have conducted our experiments at a software level, this hybrid algorithm is aimed at an hardware implementation. However, since the step we

introduce involves multiplications (and sometimes divisions), a fast multiplier has to be used to preserve the gain in terms of time. Nevertheless, if a superfast multiplier is used, then the computation of elementary functions may be performed faster using approximations by low degree polynomials on small subintervals, since the coefficients of these polynomials and the bounds on the subintervals can be tabulated. A closer study should be performed in order to decide which method is the fastest with a given multiplier.

References

1. K. Atkinson, *An introduction to numerical analysis*, 2nd edition, Wiley, 1989.
2. A. Avizienis, *Signed-digit representations for fast parallel arithmetic*, IRE Transactions on electronic computers, 10:389–400. Reprinted in E.E. Swartzlander, *Computer arithmetic*, vol. 2, IEEE Computer Society Press, 1990.
3. R.P. Brent, *Fast multiple precision evaluation of elementary functions*, Journal of the ACM, 23:242–251, 1976.
4. W.J. Cody and W. Waite, *Software manual for the elementary functions*, Prentice Hall, 1980.
5. W.J. Cody, *Implementation and testing of function software*, In *Problems and methodologies in mathematical software production*, Lecture Notes in Computer Science 142, Springer-Verlag, 1982.
6. M. Daumas, C. Mazenc, X. Merrheim and J.-M. Muller, *Modular range reduction: a new algorithm for fast and accurate computation of the elementary functions*, Journal of Universal Computer Science, 1(3):162–175, 1995.
7. M.D. Ercegovac and T. Land, *Division and square-root: digit-recurrence algorithms and implementations*, Kluwer, 1994.
8. J.-M. Muller, *Arithmétique des ordinateurs* (in french), Masson, 1989.
9. J.M. Muller, *Elementary functions*, Birkhaüser, 1997.
10. B. Parahmi, *Generalized signed-digit number systems: a unifying framework for redundant number representations*, IEEE Transactions on Computers, 39(1):89–98, 1990.
11. M. Payne and R. Hanek, *Radian reduction for trigonometric functions*, SIGNUM Newsletter, 18:19–24, 1983.
12. N. Revol and J.-C. Yakoubsohn, *Accelerated Shift-and-Add algorithms*, Research Report, extended version, `ftp://ano.univ-lille1.fr/pub/1999/ano395.ps.Z`.
13. N. Takagi, T. Asada and S. Yajima, *Redundant CORDIC methods with a constant scale factor*, IEEE Transactions on Computers, 40(9):989–995, 1991.
14. J. Volder, *The CORDIC computing technique*, IRE Transactions on electronic computers, 14–17, 1959. Reprinted in E.E. Swartzlander, *Computer arithmetic*, vol. 1, IEEE Computer Society Press, 1990.