Two Algorithms for Root Finding in Exact Real Arithmetic

Abbas Edalat *   Fabien Rico †

Abstract

We present two algorithms for computing the root, or equivalently the fixed point, of a function in exact real arithmetic. The first algorithm uses the iteration of the expression tree representing the function in real arithmetic based on linear fractional transformations and exact floating point. The second and more general algorithm is based on a trisection of intervals and can be compared with the well-known bisection method in numerical analysis. It can be applied to any representation for exact real numbers; here it is described for the sign binary system in $[-1, 1]$ which is equivalent to the exact floating point with linear fractional transformations.

Keywords : Shrinking intervals, Normal products, Exact floating point, Expression trees, Sign Binary System, Iterative method, Trisection.

1 Introduction

In the past few years, continued fractions and linear fractional transformations (lft), also called homographies or Möbius transformations, have been used to develop various frameworks for exact real arithmetic. In [17], Vuillemin proposed a representation of computable real numbers by redundant continued fractions and, using the earlier work of Gosper [8], presented various incremental algorithms for basic arithmetic operations and some transcendental functions. This framework for exact real arithmetic is closely linked with lft's; it has been further developed and implemented by Lester [10]. Nielsen and Kornerup [12] developed a general framework for representing a real number as an infinite product of matrices or as an infinite composition of lft's. De Gianantonio [3, 4] and Escardó [7] studied extensions of the programming language PCF (Programming language for Computable Functions) with a representation of real numbers by infinite composition of affine maps.

In search of a simple and efficient representation, a framework for exact real numbers using lft's has been proposed in [15, 5] which is based on the special

*Department of Computing, Imperial College, 180 Queen's Gate, London SW7 2BZ. E-mail: ae@doc.ic.ac.uk

†LIP, ENS Lyon, 46 allée d'Italie 69007 Lyon. E-mail: frico@ens-lyon.fr

base interval $[0, \infty]$ in the one-point compactification $\mathbb{R}^*$ of the real line $\mathbb{R}$. A real number is expressed by a (finite or infinite) product of integer matrices representing lft's, where the first so-called sign matrix has integer entries and the subsequent matrices, called digit matrices, have non-negative integers. This product corresponds to a shrinking sequence of rational intervals, giving better and better approximations to the real number, which are generated by applying the composition of the lft's to the base interval. A special representation where the number of sign matrices is restricted to four and that of digit matrices to three has been called exact floating point which unifies the sign binary system and the lft framework. Real valued functions are represented by so-called expression trees of lft's whose nodes have lft's with zero, one or two arguments. A complete set of efficient and on-line algorithms for computing elementary functions in this exact floating point representation has been developed by Potts [15, 14] and implemented in the functional programming language Caml. Furthermore, following the work in [3, 7], an extension of the programming language PCF with a real number data type, based on this framework, has been developed [13, 6] which is computationally adequate, i.e. given any program denoting a real number, the reduction rules of the language give arbitrarily close approximations to the real number by rational intervals.

In this paper, we deal with the problem of computing the fixed point and hence the root of a function in the above framework. It is known in computable analysis that an isolated root of a computable function is computable [18, page 496]. However, whereas several techniques are known to compute roots of a function in interval analysis [1, chapter 7], up to now the only method proposed in exact arithmetic for computing fixed points has been the feedback loop technique as in [19, 17] which can only be applied to very simple functions.

Here, we present two new algorithms for computing fixed points in exact real arithmetic. The first uses an iterative method and its idea is based on the convergence of iterates of a function near an attracting fixed point as in dynamical systems (see for example [2]). More specifically, it uses the iterates of an expression tree representing the function which also gives the canonical extension of the function to intervals. This method is a generalization of the feedback loop technique in [17].

The second algorithm uses a trisection of intervals and can be compared with the well-known bisection method in numerical computation [16]. In floating point computation, one can compare two numbers and hence use the bisection method. In contrast, in exact real arithmetic, comparison of a real number with, say, a rational number is undecidable, hence the bisection method fails in general. However comparison of a real number with two distinct rational numbers always leads to at least one successful result in finite time. This gives the underlying idea of the trisection method for computing fixed points. Any algorithm for such a double comparison has to use a concrete representation of exact real numbers [11]; here we will use the binary sign system in $[-1, 1]$ which is equivalent with the lft representation in [15, 5]. However, the trisection algorithm can be used with any other representation for exact real numbers.

In section 2, we recall the background work for this paper and summarize the basic ideas for representing real numbers by exact floating point and representing functions by expression trees which are essential for presenting our algorithms. In section 3, we provide sufficient conditions for an expression tree to induce the canonical extension of the real function to intervals. In section 4, the iterative algorithm to compute the fixed point of a function is presented. Finally, in section 5, we introduce the trisection algorithm and describe its implementation.

2 The representation of real numbers and real functions

2.1 Representation of real numbers

Representation of real numbers by infinite composition of linear fractional transformations were presented in [17] and [12]. We will briefly review the representation of real numbers by linear fractional transformations based on the special base interval $[0, \infty]$ introduced in [15, 5]. As in [17], we will work in the one-point compactification of $\mathbb{R}$ denoted by $\mathbb{R}^*$. A real number is represented by the intersection of a sequence of shrinking rational intervals which are encoded, or generated by linear fractional transformations applied to $[0, \infty]$. We will explain this below and show how the interval $[0, \infty] \subset \mathbb{R}^*$ plays a crucial role in the framework.

Let

$$\mathbb{V} = \{\begin{pmatrix} a \\ b \end{pmatrix} | a, b \in \mathbb{Z}\}, \qquad \mathbb{M} = \{\begin{pmatrix} a & c \\ b & d \end{pmatrix} | a, b, c, d \in \mathbb{Z}\}$$

$$\mathbb{T} = \{\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix} | a, b, c, d, e, f, g, h, \in \mathbb{Z}\}$$

be, respectively, the sets of vectors, matrices and (rank 3) tensors with integer coefficients.

Vectors, matrices and tensors induce **linear fractional transformations (lft)** defined by:

Definition 2.1 (Linear fractional transformation) *A 0-dimensional lft is a fraction in $\mathbb{R}^*$ which is obtained by a vector:*

$$\Theta\begin{pmatrix} a \\ b \end{pmatrix} = \frac{a}{b}. \tag{1}$$

A 1-dimensional lft is a function from $\mathbb{R}^*$ to $\mathbb{R}^*$ which is obtained from a matrix:

$$\Theta\begin{pmatrix} a & c \\ b & d \end{pmatrix}(x) = \frac{ax + c}{bx + d}. \tag{2}$$

A 2-dimensional lft is a function from $\mathbb{R}^* \times \mathbb{R}^*$ to $\mathbb{R}^*$ which is obtained form a tensor:

$$\Theta\begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}(x, y) = \frac{axy + cx + ey + g}{bxy + dx + fy + h}. \tag{3}$$

We identify any lft with the corresponding vector, matrix or tensor, i.e. we write the vector, matrix or tensor K for the lft $\Theta(K)$. This correspondence is unique up to multiplication by a non-negative integer. The composition of 1-dimensional lft's is equivalent to matrix multiplication. Given two non-trivial rational intervals $[p, q]$ and $[r, s]$ with $p \neq q$ and $r \neq s$, there exists an lft $\phi \in \mathbb{M}$ with $\phi([p, q]) = [r, s]$. It follows that if we fix a base interval, then we can express, or encode, all other non-trivial intervals as the image of this base interval under an lft. The most efficient base interval is $[0, \infty]$ as no computation is needed to determine the lft in the proposition. Given, for example, a rational interval $[\frac{a}{b}, \frac{c}{d}] \subset \mathbb{R}$ with $b, d > 0$ and $ad - bc > 0$, the maps $x \mapsto \frac{ax+c}{bx+d}$ and $x \mapsto \frac{cx+a}{dx+b}$ have integer coefficients and map $[0, \infty]$ onto $[\frac{a}{b}, \frac{c}{d}]$ respectively reversing and preserving the orientation. We therefore fix our base interval to be $[0, \infty]$ and define the following.

Definition 2.2 *The information,* $\mathbf{Info}(K)$*, given by an lft* K *is an interval of* $\mathbb{R}^*$ *defined by:*

$$\mathbf{Info}(V) = \{V\} \qquad \mathbf{Info}(M) = M([0, \infty]) \qquad \mathbf{Info}(T) = T([0, \infty], [0, \infty]).$$

Let $\mathbb{V}^+ \subset \mathbb{V}$, $\mathbb{M}^+ \subset \mathbb{M}$ and $\mathbb{T}^+ \subset \mathbb{T}$ be the subsets of vectors, matrices and tensors K with $\mathbf{Info}(K) \subseteq [0, \infty]$. Note that these subsets are precisely the lft's whose entries are all non-negative or non-positive and do not contain a $\begin{pmatrix} 0 \\ 0 \end{pmatrix}$ column.

For all matrices $M, N \in \mathbb{M}$ we have: $N \in \mathbb{M}^+$ iff $\mathbf{Info}(MN) \subseteq \mathbf{Info}(M)$; i.e. right multiplication by a non-negative matrix corresponds to the refinement of information. Other notions of refinement of intervals can be found in [12, 7].

Definition 2.3 *A* **signed normal product (snp)** *and an* **unsigned normal product (unp)** *are defined recursively by :*

$$\begin{array}{lcll} \mathit{snp} & ::= & V | M : \mathit{unp} & \text{where } \alpha \in \mathbb{V},\ M \in \mathbb{M} \\ \mathit{unp} & ::= & V | M : \mathit{unp} & \text{where } V \in \mathbb{V}^+,\ M \in \mathbb{M}^+ \end{array}$$

So for every snp (respectively, unp) $M_1 M_2 \cdots$, the information given by the finite product $M_1 \cdots M_n$ gives a sequence of shrinking nested rational intervals in $\mathbb{R}^*$ (respectively, in $[0, \infty]$). One can use normal products in order to represent real numbers (see [5]).

Note that a snp $M_1 M_2 \cdots$ is an unp $M_2 M_3 \cdots$ preceded by a matrix with integer coefficients. By analogy with the usual representation of a real number, we call M_1 the sign matrix; the others are called digit matrices. In the following, we use tensors, matrices and vectors with non-negative coefficients except for the sign matrix of a snp.

In a general snp, the sizes of integers grow unjustifiably large with matrix multiplication, disproportionate with the length of the intervals that the snp represents. Furthermore, information is refined essentially in an arbitrary manner which means that we cannot compute the rate of convergence of information. These problems can be overcome by restricting the class of matrices in a snp to a suitable finite set. **Exact floating point** [5] in base 2 is a snp consisting of the following matrices:

$$
\begin{aligned}
S_0 &= \begin{pmatrix} 1 & -1 \\ 1 & 1 \end{pmatrix} \\
S_- &= \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \\
S_\infty &= \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \\
S_+ &= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \\
D_1 &= \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \\
D_0 &= \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix} \\
D_{-1} &= \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}
\end{aligned}
$$

The sign matrices S_0, S_-, S_∞, S_+ form a cyclic group of rotations of $\mathbb{R}^*$ identified as the unit circle in the plane via the stereographic map. The group is generated by the lft S_0 which represents clockwise rotation by $\pi/2$; it is the only finite cyclic group of rotations of $\mathbb{R}^*$ which has a representation by matrices with integer coefficients [5]. We have $\mathbf{Info}(S_+) = [0,\infty]$, $\mathbf{Info}(S_\infty) = [-1,1]$, $\mathbf{Info}(S_-) = [\infty,0]$, $\mathbf{Info}(S_0) = [-1,1]$. The digit matrices are obtained as $D_i = S_\infty d_i S_0$ where $d_i : x \mapsto \frac{x+i}{2} : \mathbb{R}^* \to \mathbb{R}^*$. Note that $S_\infty = S_0^{-1}$ and the maps d_i are the affine maps which generate the binary sign system in $[-1,1]$. Furthermore, the normal product $S_0 D_{i_1} D_{i_2} \cdots$ corresponds to a real number in $[-1,1]$ whose binary sign expansion is given by $i_1 i_2 \cdots$. In fact, for each $n > 0$,

$$
\begin{aligned}
S_0 D_{i_1} D_{i_2} \cdots D_{i_n}[0,\infty] &= (S_0 D_{i_1} S_\infty)(S_0 D_{i_2} S_\infty) \cdots (S_0 D_{i_n} S_\infty)(S_0[0,\infty]) \\
&= d_{i_1} d_{i_2} \cdots d_{i_n}[-1,1].
\end{aligned}
$$

Our assertion now follows by taking the intersection of these intervals.

We have $\mathbf{Info}(D_1) = [1,\infty]$, $\mathbf{Info}(D_0) = [1/3,3]$ and $\mathbf{Info}(D_{-1}) = [0,1]$. The three digit matrices above have also been used by Nielsen and Kornerup [12]. The digit matrices D_1, D_0 and D_{-1} contract distances in $[0,\infty]$ by a factor $1/2$ with respect to the metric $d(x,y) = |S_0(x) - S_0(y)| = |\frac{x-1}{x+1} - \frac{y-1}{y+1}|$. Consequently, in exact floating point information is refined at a steady rate. Any real number can be expressed in exact floating point, since the information in the four sign matrices (respectively, the three digit matrices) above overlap and cover $\mathbb{R}^*$ (respectively, $[0,\infty]$), i.e. these seven matrices provide a redundant representation of real numbers. For example, an unp for $\sqrt{2}$ is given by [15]

$$\sqrt{2} = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}^\omega = \begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} 1 & 2 \\ 1 & 1 \end{pmatrix}\cdots.$$

This can be converted to exact floating point. The result is, due to redundancy of the representation, not unique. For example, we can have:

$$\sqrt{2} = D_0 D_1 D_{-1} D_1 D_0 D_{-1} D_0 D_0 D_0 D_0 D_{-1} \cdots$$

which so far gives the rational interval $[\frac{1199}{849}, \frac{1200}{848}]$ as an approximation to $\sqrt{2}$.

2.2 Representation of functions

We can represent rational functions with tensors. In fact the basic arithmetic operations are given, as originally in the work of Gosper [8], by 2-dimensional lft's.

$$\begin{aligned} \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x,y) &= x+y \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}(x,y) &= x\times y \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}(x,y) &= x\div y \end{aligned}$$

Signed and unsigned normal products are generalized to singed and unsigned expression trees by allowing tensors as follows.

Definition 2.4 *A* **signed expression tree (sext)** *and an* **unsigned expression tree (uext)** *are binary trees defined by:*

$$\begin{array}{rcll} sext & ::= & V|M(uext)|T(uext,uext) & \text{where } V\in\mathbb{V},\ M\in\mathbb{M},\ T\in\mathbb{T} \\ uext & ::= & V|M(uext)|T(uext,uext) & \text{where } V\in\mathbb{V}^{+},\ M\in\mathbb{M}^{+},\ T\in\mathbb{T}^{+} \end{array}$$

A finite truncation of such an expression tree corresponds to a finite subtree such that the removed nodes are replaced by the base interval $[0,\infty]$. Any such truncation denotes the composition of a finite number of lft's applied to $[0,\infty]$. Hence, it denotes a compact interval. The set of all compact intervals obtained as such is filtered and their intersection gives the value of the expression tree.

In order to compute the value of an expression tree, we need a way to transform such a tree into a snp. We first need the rules for composition of lft's of different dimensions. We say that a tensor T is composed of two matrices $T=(T_0,T_1)$, and a matrix M of two vectors $M=(M_0,M_1)$.

Definition 2.5 *Let the dot product, left product, and the right product, denoted respectively by* $\cdot$, Ⓛ *and* Ⓡ *, be defined by:*

$$\begin{aligned} (M\cdot V)_i &= \sum_{j=0,1} M_{ij}V_j \\ (M\cdot N) &= (M\cdot N_0, M\cdot N_1) \\ (M\cdot T) &= (M\cdot T_0, M\cdot T_1) \\ T \circledR V &= (T_0\cdot V, T_1\cdot V) \\ T \circledR M &= (T_0\cdot M, T_1\cdot M) \\ T \circled{L} V &= T^T \circledR V \\ T \circled{L} M &= (T^T \circledR M)^T \end{aligned}$$

where T^{t} *is the* **transpose** *of* T *obtained by swapping the two middle columns of* T *i.e. if* $T = \begin{pmatrix} a & c & e & g \\ b & d & f & h \end{pmatrix}$ *then* $T^{\mathrm{t}} = \begin{pmatrix} a & e & c & g \\ b & f & d & h \end{pmatrix}$.

Let $\mathcal{T} = T(\mathcal{T}_1, \mathcal{T}_2)$ be a signed expression tree having the tensor T as its root node with left and right unsinged expression subtrees $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. In order to obtain the sign matrix of the snp, we choose M such that $M([0,\infty]) \supseteq T([0,\infty],[0,\infty])$. Then, M is the sign matrix of the snp. We now replace T in $\mathcal{T}$ by $M^{-1} \cdot T$ which has non-negative coefficients because $M([0,\infty]) \supseteq T([0,\infty],[0,\infty])$. This is called **emission**. In order to have a more precise result, we absorb some information in the tensor T. That means emitting matrices M_1 and M_2 from $\mathcal{T}_1$ and $\mathcal{T}_2$ and replacing T with $T \textcircled{L} M_1 \textcircled{R} M_2$. This gives an incremental algorithm to compute an expression tree.

We can also represent all basic elementary functions such as $f = \tan, \arctan, \log$ and $\exp$ in terms of expression trees with an entry x of the form given in figure 1; see [15, 14]. Let x be a real number. The value $f(x)$ of the infinite expression tree in the figure is given in terms of the truncated expression trees $S_n^f(x)$ by $f(x) = \bigcap_n S_n^f(x)$, which in general may be a compact interval. Here, we have: $S_n^f(x) = \{T_1(x, T_2(x, , \cdots, T_n(y,z)\cdots))|y,z \in [0,\infty]\}$.

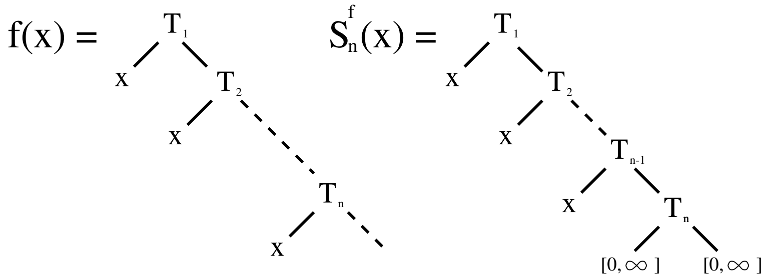

Figure 1: The function f represented by an expression tree

For example, the function arctan has the following continued fraction expansion:

$$\arctan x = \cfrac{x}{1 + \cfrac{\frac{x^2}{3}}{1 + \cfrac{\frac{4x^2}{15}}{1 + \ddots}}},$$

which can be transformed [15, 14] into

$$\arctan x = \prod_{n=1}^{\infty} \begin{pmatrix} 0 & x \\ n^2 x & 2n-1 \end{pmatrix}.$$

This is an infinite composition of lft's with non-negative coefficients but now each lft has x as a parameter, i.e., it is a function of two arguments. This can be rewritten as an infinite expression tree:

$$\arctan x = \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \end{pmatrix} \left[x, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 4 & 0 & 0 & 3 \end{pmatrix} \left[x, \begin{pmatrix} 0 & 1 & 0 & 0 \\ 9 & 0 & 0 & 5 \end{pmatrix} [x, \cdots] \right] \right]$$

We need an algorithm to compute the value of an expression tree $\mathcal{T}(x)$ when the entry x corresponds to an extended real number represented by an unp $D_1 D_2 \cdots$. Let $\mathcal{T}_n(x)$ denote the **truncation** of the $\mathcal{T}(x)$ at depth n, i.e. any node with n edges away from the root node is replaced by $[0, \infty]$. In the case of the expression tree in figure 1, $\mathcal{T}_n(x)$ is simply $S_n^f(x)$. We let $(x)_n$ denote the truncation at depth n of the tree representing x. In the case of an unp, $(D_1 D_2 \cdots)_n = D_1 \cdots D_n[0, \infty]$. We can compute the information given by the finite tree $\mathcal{T}_n((x)_n)$ using the absorption and emission rules we have defined. The sequence $\langle \mathcal{T}_n((x)_n) \rangle_{n \in \mathbb{N}}$ is a nested shrinking sequence of compact intervals which tends to the value of the expression tree $\mathcal{T}(D_1 D_2 \cdots)$. Note that the algorithm is incremental, i.e. it is not necessary to recompute $\mathcal{T}_n((x)_n)$ in order to evaluate $\mathcal{T}_{n+1}((x)_{n+1})$; see [6] where a deterministic algorithm for evaluation of an expression tree is given.

3 Canonical extension of functions

Let f be a function from $[0, \infty]$ to $\mathbb{R}^*$ such that $f(x)$ is represented by an expression tree $\mathcal{T}(x)$. A special case is provided by the expression tree in figure 1. The canonical extension of f to intervals, which we simply denote by f, is given by $f[a, b] = \{f(x) | x \in [a, b]\}$. However, the expression tree $\mathcal{T}(x)$ also gives us an extension $\hat{f}$ of f to intervals defined by $\hat{f}([a, b]) = \bigcap_n \mathcal{T}_n([a, b])$, with

$$\mathcal{T}_n([a, b]) = \{\mathcal{T}_n(x_1, \cdots, x_m; y_1, \cdots, y_k) | x_i \in [a, b], y_j \in [0, \infty]\} \tag{4}$$

where x_i $(i = 1, \cdots, m)$ are the distinct instances of x and y_j $(j = 1, \cdots, k)$ are variables at the nodes replaced by $[0, \infty]$. Here, $[a, b]$ can be represented by an unp. We note that, for $x \in [0, \infty]$ represented by an unp, it is easily shown that $\hat{f}(\{x\}) = \{f(x)\}$. Moreover, $\hat{f}$ is monotonic, i.e. $\hat{f}([a, b]) \subseteq \hat{f}([c, d])$ if $[a, b] \subseteq [c, d]$.

If $\hat{f}$ is represented by a tree $\mathcal{T}$ and $[a, b]$ by an unp, the the value of $\hat{f}([a, b])$ is computed in the same way as for an extended real i.e. by computing incrementally the finite trees $\mathcal{T}_n(([a, b])_n)$ where $([a, b])_n$ is the truncation of the representation of $[a, b]$ at the depth n.

In some cases, the expression tree for a function f gives us the canonical extension of f to intervals, i.e. $f[a, b] = \hat{f}([a, b])$ for any interval $[a, b] \subseteq [0, \infty]$. This property is used in the first method in section 4. We will always use the canonical extensions of a matrix or a tensor to intervals. Here, we will give sufficient conditions for an expression tree to represent the canonical extension of a function on the real line. We first need some definitions.

A tensor $T : [0, \infty] \times [0, \infty] \to (-\infty, \infty)$ is said to be

- **left-increasing** if $\forall y \in [0, \infty]$, $x \mapsto T(x, y)$ is increasing.
- **left-decreasing** if $\forall y \in [0, \infty]$, $x \mapsto T(x, y)$ is decreasing.
- **right-increasing** if $\forall x \in [0, \infty]$, $y \mapsto T(x, y)$ is increasing.
- **right-decreasing** if $\forall x \in [0, \infty]$, $y \mapsto T(x, y)$ is decreasing.
- **monotonic** if it is
  - left-increasing or left-decreasing, and,
  - right-increasing or right-decreasing.

A matrix $M : [0, \infty] \to (-\infty, +\infty)$ is said to be **increasing**, respectively **decreasing**, if the function $x \mapsto M(x)$ is increasing, respectively decreasing.

Let $\mathcal{T}(x)$ be a general expression tree with x an extended real number represented by an unsigned expression tree. Assume $x_1, \cdots, x_n, \cdots$ are the difference instances of x; this list can be either finite or infinite. If all tensors in $\mathcal{T}(x)$ are monotonic, then $\mathcal{T}(x_1, x_2, \cdots)$ is monotonic in each x_i. In fact for all $i \geq 1$, there exists a unique path $(N_1, d_1), \cdots, (N_m, d_m)$ from the root of $\mathcal{T}(x)$ to x_i, where m is the distance of the root to x_i. Here, N_j is a node of the tree (a tensor or a matrix) and d_j is the direction to the next node of the path i.e. d_j is $left$ or $right$ if N_j is a tensor and *next* if it is a matrix. Then, the monotonicity at x_j is determined by its associated path. In fact, let $\varepsilon_1, \cdots, \varepsilon_m \in \{-1, 1\}$ such that

$$\varepsilon_j = \begin{cases} 1 & \text{if } N_j \text{ is an increasing matrix} \\ -1 & \text{if } N_j \text{ is a decreasing matrix} \\ 1 & \text{if } N_j \text{ is a left-increasing tensor} \ \text{ and } d_j = left \\ -1 & \text{if } N_j \text{ is a left-decreasing tensor} \ \text{ and } d_j = left \\ 1 & \text{if } N_j \text{ is a right-increasing tensor} \ \text{ and } d_j = right \\ -1 & \text{if } N_j \text{ is a right-decreasing tensor} \ \text{ and } d_j = right \end{cases}$$

Then, $\mathcal{T}(x_1, \cdots, x_i, \cdots)$ is increasing in x_i if $\prod_{j=1}^{m} \varepsilon_j = 1$, and is decreasing in x_i if $\prod_{j=1}^{m} \varepsilon_j = -1$.

Definition 3.1 *$\mathcal{T}$ is consistently monotonic if it has the same monotonicity for all x_i ($i \geq 0$).*

Theorem 3.2 *Let $f : [0, \infty] \to \mathbb{R}^*$ be a continuous function represented by a consistently monotonic expression tree. Then f is monotonic and the extension $\hat{f}$ of f to intervals given by the expression tree is the canonical extension of f.*

Proof

Since, every tensor in the truncated tree $\mathcal{T}_n$ is monotonic, the function $(x_1, x_2, \cdots, x_m; y_1, \cdots, y_k) \mapsto \mathcal{T}_n(x_1, x_2, \cdots, x_m; y_1, \cdots, y_k)$ as in Equation 4 is monotonic in each x_i and y_j. The second condition for a consistently monotonic expression tree implies that this monotonicity is the same for all x_i. This means that for all $[a,b] \subseteq [0,\infty]$, $min(\hat{f}([a,b])) = min(\hat{f}(\{a\}) \cup \hat{f}(\{b\}))$ and $max(\hat{f}([a,b])) = max(\hat{f}(\{a\}) \cup \hat{f}(\{b\}))$. We have $\hat{f}(\{a\}) = \{f(a)\}$ and $\hat{f}(\{b\}) = \{f(b)\}$. Furthermore, $[min(f(a), f(b)), max(f(a), f(b))] \subseteq f([a,b])$ because f is continuous. Hence,

$$\begin{aligned}
\hat{f}[a,b] &= [min(\hat{f}[a,b]), max(\hat{f}[a,b])] \\
&= [min(\hat{f}(\{a\}) \cup \hat{f}(\{b\})), max(\hat{f}(\{a\}) \cup \hat{f}(\{b\}))] \\
&= [min(f(a), f(b)), max(f(a), f(b))] \\
&\subseteq f([a,b]).
\end{aligned}$$

As $f([a,b]) \subseteq \hat{f}([a,b])$, it follows that $f([a,b]) = \hat{f}([a,b])$. Furthermore since, for all intervals $[a,b]$, we have $f([a,b]) = [f(a), f(b)]$ or $f([a,b]) = [f(b), f(a)]$, we conclude that f is monotonic. □

We give some examples. In the following, we will only deal with functions from $[0,\infty]$ to $[0,\infty]$.

Example 3.3 *Let* $T = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ *and* $T' = \begin{pmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$. *Then, T is left-increasing and right-decreasing and T' is left-increasing and right-increasing. We have $T(x,x) = T'(x,x) = x+1$. Since the identity function is increasing, the tree $T'(x,x)$ is consistently monotonic whereas $T(x,x)$ is not. It follows that $T'(x,x)$ is the canonical extension of the function $x \mapsto x+1$. Furthermore, $T(x,x)$ is not the canonical extension of this function as* $\mathbf{Info}(T) = T([0,\infty],[0,\infty]) = [0,\infty] \neq [1,\infty] = \mathbf{Info}(T')$.

Example 3.4 *Let T be the tensor* $\begin{pmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$. *Then T is left-increasing and right-decreasing. Let i be the lft $x \mapsto 1/x$. Note that i is decreasing. Then, $T(x, i(x))$ represents the canonical extension of the function $x \mapsto \frac{x^2+x}{2x+1}$. On the other hand, $T(x, Id(x))$ represents the lft $f : x \mapsto \frac{x+1}{x+2}$ but not its canonical extension, since $T([0,\infty],[0,\infty]) = [0,\infty]$ whereas* $\mathbf{Info}(f) = [\frac{1}{2}, 1]$.

Example 3.5 *For an example of an infinite expression tree, consider the expression tree for $\sqrt{x}$ [14] given by $g(x) = T(x, T(x, \cdots))$ where*

$$T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix}$$

As T is left-increasing and right-increasing, g is consistently monotonic and is therefore the canonical extension of $x \mapsto \sqrt{x}$.

4 The iterative method

Our first algorithm is an iterative method to find the attracting fixed point of a function by using an expression tree which gives the canonical extension of the function to intervals. Recall that any root of a function f is a fixed point of the function $g = f + \mathrm{Id}$ where Id is the identity function. Therefore, this algorithm can be used to compute roots of functions as well.

We first present an algorithm which converts a nested shrinking sequence of non-negative rational compact intervals, which converges to a real number, into an exact floating point. Suppose $\langle I_n \rangle_{n\in\mathbb{N}}$ is such a sequence. Then the digit matrices in the exact floating point are obtained recursively by the following conversion scheme.

The conversion algorithm:

- Let inverse $= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and i=0
- repeat
  - while $\forall D \in \{D_{-1}, D_0, D_1\}$. inverse $\cdot\, I_i \not\subseteq D$ do $i = i + 1$
  - let D be this digit and put inverse $= D^{-1} \cdot$ inverse.

We say that a function $f : \mathbb{R} \to \mathbb{R}$ has an **attracting fixed point** x in $[a, b]$ if $\bigcap_{n\in\mathbb{N}} f^n[a, b] = \{x\}$. Let $f : [0, \infty] \to [0, \infty]$ be a function with an attracting fixed point x in $[0, \infty]$. Suppose that there exists an unsigned expression tree $x \mapsto \mathcal{T}(x)$ which gives the canonical extension of f to intervals.

We will construct a nested shrinking sequence of rational intervals converging to the fixed point of f. The following simple result from elementary analysis is needed in our proofs.

Lemma 4.1 *Let I_{mn} be a double sequence of compact real intervals such that for each fixed n the sequence $\langle I_{mn}\rangle_{m\in\mathbb{N}}$ is nested and shrinking and for each fixed m the sequence $\langle I_{mn}\rangle_{n\in\mathbb{N}}$ is nested and shrinking. Then,*

$$\bigcap_{m,n\in\mathbb{N}} I_{mn} = \bigcap_{m\in\mathbb{N}} \bigcap_{n\in\mathbb{N}} I_{mn} = \bigcap_{n\in\mathbb{N}} \bigcap_{m\in\mathbb{N}} I_{mn} = \bigcap_{m\in\mathbb{N}} I_{mm}.$$

Proposition 4.2 *Let $K_{nm}(x)$ be the tree defined by*

$$K_{nm}(x) = (\mathcal{T}_m)^n(x)$$

where $\mathcal{T}_n(x)$ is the truncation of $\mathcal{T}(x)$ at depth n. Then,

$$\bigcap_{n\in\mathbb{N}} K_{nn}([0,\infty]) = \bigcap_{n,m\in\mathbb{N}} K_{nm}([0,\infty]) = \{x\}$$

$\mathcal{P}roof$

Let $I \subseteq [0, \infty]$ be an interval. For fixed n, we have that $\langle K_{nm}(I)\rangle_{m\in\mathbb{N}}$ is a nested shrinking sequence of compact intervals. Therefore, the double sequence of compact intervals $\mathcal{T}_{m'}(K_{nm}(I))$ is nested and shrinking in m for fixed m' and in m' for fixed m. We show by induction that for every $n \in \mathbb{N}$

$$\bigcap_{m\in\mathbb{N}} K_{nm}(I) = f^n(I).$$

The interval $K_{1m}(I)$ is the approximation up to level m of $f(I)$, since the tree corresponds to the canonical extension of f. This proves the base case. Assume inductively that

$$\bigcap_{m\in\mathbb{N}} K_{nm}(I) = f^n(I).$$

Then by lemma 4.1 we have

$$\bigcap_{m} K_{n+1m}(I) = \bigcap_{m'}\bigcap_{m} \mathcal{T}_{m'}(K_{nm}(I)) = \bigcap_{m'} \mathcal{T}_{m'}(f^n(I)) = f^{n+1}(I),$$

which completes the inductive proof. Since

$$\bigcap_{n\in\mathbb{N}} f^n([0,\infty]) = \{x\}$$

and $K_{nm}([0,\infty])$ is nested and shrinking in m for n fixed and nested and shrinking in n for m fixed, the result follows by lemma 4.1. □

Using $I_n = K_{nn}([0,\infty])$ in the conversion algorithm presented in the beginning of this section, we can obtain the fixed point x in exact floating point. We have, however, no inductive algorithm to compute $K_{n+1,n+1}[0,\infty]$ from $K_{nn}[0,\infty]$. This means that we have no efficient method to evaluate $K_{nn}[0,\infty]$. Therefore, we seek another tree which can be inductively computed. Let $L_1 = \mathcal{T}_1$ and $L_n = \mathcal{T}_n(L_{n-1})$.

Figure 4 shows the construction of L_n when the expression tree is of the special form $\mathcal{T}(x) = T_1(x, T_2(x, T_3(\cdots)))$.

Lemma 4.3

$$\bigcap_{n\in\mathbb{N}} L_n = \{x\}.$$

$\mathcal{P}roof$

It follows by a simple induction that L_n is a nested shrinking sequence of compact intervals; in fact the tree represented by L_n is a subtree of that represented by L_{n+1} for all $n \geq 1$. Therefore, the intersection $\bigcap_{n\in\mathbb{N}} L_n$ is a non-empty compact interval. The information in every expression tree with tensors in $\mathbb{T}^+$ and matrices in $\mathbb{M}^+$ is included in $[0,\infty]$. So it is easy to show by induction that $\forall n \geq 1.\ L_{2n} \subseteq K_{nn}([0,\infty])$. Therefore,

$$\bigcap_{n\in\mathbb{N}} L_n \subseteq \bigcap_{n\in\mathbb{N}} K_{nn} = \{x\}$$

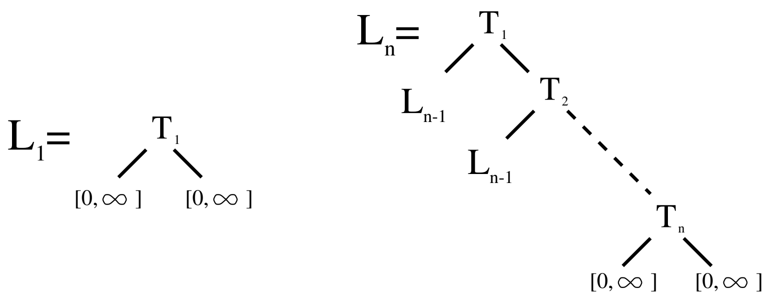

Figure 2: L_n by induction

and the result follows. $\square$

Compared with $I_n = K_{nn}([0,\infty])$, the conversion algorithm with $I_n = L_n$ provides a more efficient method of computing the fixed point x. Note that if the expression tree consists of a single matrix M, then we have $K_{nn} = L_n = M^n$. In this case, our algorithm will be essentially the feedback algorithm presented in [17] but adapted to the exact floating point format. On the other hand, if the expression tree for f has more than one branch, then the computation of L_n will be in general exponential in n. For example, for the special expression tree in figure 4, L_n has more than 2^n nodes.

Next we will generalize the above result to any compact interval $[a,b] \subseteq \mathbb{R}^*$.

Lemma 4.4 *Let M be a non-singular matrix with integer coefficients. Let f be a function, such that $f(M([0,\infty])) \subseteq M([0,\infty])$. If $x \in M([0,\infty])$ is a fixed point of f then $M^{-1}fM$ is a function from $[0,\infty]$ to $[0,\infty]$ which has a fixed point y such that $x = My$. If x is an attracting fixed point for f in $M([0,\infty])$, then y is an attracting fixed point for $M^{-1}fM$ in $[0,\infty]$.*

Proof

Let $y = M^{-1}(x)$, then $f(x) = x \iff M^{-1}f(My) = y$. Furthermore, if

$$\bigcap_{n\in\mathbb{N}} f^n(M([0,\infty])) = \{x\}$$

then, from $(M^{-1}fM)^n = M^{-1}f^nM$, we obtain

$$\bigcap_{n\in\mathbb{N}} (M^{-1}fM)^n([0,\infty]) = M^{-1} \bigcap_{n\in\mathbb{N}} f^n(M([0,\infty])) = \{y\}.$$

$\square$

Therefore, if $f : M[0,\infty] \to M[0,\infty]$ has an attracting fixed point in $M[0,\infty]$, we can find, using a canonical expression tree for $M^{-1}fM$, an exact floating point for the fixed point y of $M^{-1}fM$. Then the conversion algorithm yields an exact floating point for the fixed point $x = M^{-1}y$ of f.

Under a very general condition, we can obtain an expression tree for $M^{-1}fM$ given an expression tree for f. We assume that $f(M[0,\infty])$ is contained in the interior of $M[0,\infty]$. Suppose that $f(x)$ has an unsigned expression tree $\mathcal{T}(x)$ which represents the canonical extension of f. Then, the induced tree for $M^{-1}fM$, given by $M^{-1}\mathcal{T}(Mx)$, can only have negative coefficients in the root node and it represents the canonical extension of $M^{-1}fM$. Since $f(M([0,\infty]))$ is in the interior of $M([0,\infty])$, there exists n^* such that $\mathcal{T}_{n^*}(M[0,\infty]) \subseteq M([0,\infty])$. It is then possible to reduce the expression tree $M^{-1}\mathcal{T}(Mx)$ to an unsigned expression tree which represents the canonical extension of $M^{-1}fM$. To do this, we find the least integer $n \geq 0$ such that the root node becomes non-negative after absorbing into the root node the information from all the nodes with depth less than n. Since $\mathcal{T}_{n^*}(M[0,\infty]) \subseteq M([0,\infty])$, this scheme will terminate in finite time.

The above assumptions will hold if f has an expression tree corresponding to the canonical extension and if it has a fixed point x such that $|f'(x)| < 1$. If $|f'(x)| > 1$, and if we have an expression tree for a local inverse f^{-1} of f, then we can obtain the fixed point of f by finding the fixed point of this local inverse.

Another remark is that the canonical extension is important for proving that $\bigcap_{n,m\in\mathbb{N}} K_{nm} = \{x\}$. But there are certainly other expression trees representing non-canonical extensions of f such that this property still holds.

5 The trisection method

Our second algorithm for root finding is more general and is based on comparison of real numbers with zero up to a certain precision. It is basically a trisection method which can be considered as the redundant version of the well-known bisection method. It can be adapted to any representation of exact real numbers. We will describe it here in the binary sign system, equivalently normal product of lft's in exact floating point, to compute the root of a function. As we have seen such an algorithm can also be used to compute the fixed point of a function f by computing the root of $f - \mathrm{Id}$.

Let $\mathbb{E}$ be the set of restricted signed normal products $SD_1D_2\cdots$ in exact floating point, i.e. $S \in \{S_+, S_-, S_0, S_\infty\}$ and $D_n \in \{D_{-1}, D_0, D_1\}$.

Definition 5.1 *We define a function from $\mathbb{E} \times \mathbb{N}$ to $\{+,-,\perp\}$ which gives the sign up to a given precision as follows.*

$$sign(SD_1D_2\cdots, n) = \begin{cases} + & \text{if } \forall x \in \mathbb{R} \cap SD_1\cdots D_n([0,\infty]).\ 0 \leq x \\ - & \text{if } \forall x \in \mathbb{R} \cap SD_1\cdots D_n([0,\infty]).\ x \leq 0 \\ \perp & \text{otherwise.} \end{cases}$$

5.1 The basic algorithm

Let $I \subset [-\infty,+\infty]$ be a proper subinterval of $[-\infty,+\infty]$. Let $g : I \to (-\infty,+\infty)$ be a continuous function with a unique root in (a,b). As g is continuous, there are the following four possibilities:

Definition 5.2 *We say that g or its root x is of type*

$$\begin{array}{lll} ++ & if & (\forall y \in I.\, y < x \;\Rightarrow\; g(y) > 0) \textit{ and } (\forall y \in I.\, y > x \;\Rightarrow\; g(y) > 0) \\ +- & if & (\forall y \in I.\, y < x \;\Rightarrow\; g(y) > 0) \textit{ and } (\forall y \in I.\, y > x \;\Rightarrow\; g(y) < 0) \\ -+ & if & (\forall y \in I.\, y < x \;\Rightarrow\; g(y) < 0) \textit{ and } (\forall y \in I.\, y > x \;\Rightarrow\; g(y) > 0) \\ -- & if & (\forall y \in I.\, y < x \;\Rightarrow\; g(y) < 0) \textit{ and } (\forall y \in I.\, y > x \;\Rightarrow\; g(y) < 0.) \end{array}$$

Note that a generic root is of type $-+$ or $+-$. In fact if a continuous map has a root of type $--$ or $++$, then a typical perturbation of the map will either have no roots at all or will have two roots, one of type $-+$ and the other of type $+-$. In this paper, we will only be concerned with generic roots of type $-+$ and $+-$. Let I be a closed rational interval. We can assume without loss of generality that $I = [-1, 1]$. For otherwise, we let $M \in \mathbb{M}$ be such that $M[-1, 1] = I$, then we can replace g with gM which is continuous and has a unique root $M^{-1}x \in (-1, 1)$.

Note that the type of g can be determined by computing the signs of $g(-1)$ and $g(1)$. We now describe the trisection algorithm to compute recursively the digit matrices of an exact floating point $S_0 D_{i_1} D_{i_2} \cdots D_{i_n} D_{i_{n+1}} \cdots$ for the root $x \in (-1, 1)$ of g if it is of type $+-$ or of type $-+$. This normal product immediately gives us the the binary sign expansion $x = i_1 i_2 \cdots$. We first compute $g(-\frac{1}{2})$, $g(0)$, and $g(\frac{1}{2})$ in parallel. Since there is only one fixed point, at least two of the numbers are not zero. Using the function sign, it is possible to know the sign of two of them after a finite time. Recall that $S_0 D_{-1}([0, \infty]) = [-1, 0]$, $S_0 D_0([0, \infty]) = [-\frac{1}{2}, \frac{1}{2}]$ and $S_0 D_1([0, \infty]) = [0, 1]$.

- If we know the sign of $g(0)$, we know whether $x > 0$ or $x < 0$ and so $D_{i_1} = D_{-1}$ or $D_{i_1} = D_1$.
- Otherwise, we know the sign of $c = g(-\frac{1}{2})$ and $d = g(\frac{1}{2})$ and so
  - if c and d are positive then $D_{i_1} = D_1$ when g is of type $+-$ and $D_{i_1} = D_{-1}$ when g is of type $-+$
  - if c and d are negative then $D_{i_1} = D_{-1}$ when g is of type $+-$ and $D_{i_1} = D_1$ when g is of type $-+$
  - if c and d do not have the same sign, then $D_{i_1} = D_0$.

Suppose we have obtained the n digit matrices for x i.e. $x = S_0 D_{i_1}, \cdots, D_{i_n} \cdots$. Consider the function $g_n : [-1, 1] \to (-\infty, +\infty)$ with $g_n = g S_0 D_{i_1} \cdots D_{i_n} S_\infty$. Then $g_n : [-1, 1] \to (-\infty, +\infty)$ has a unique root $y = S_0 D_{i_n}^{-1} \cdots D_{i_1}^{-1} S_\infty(x)$ in $(-1, 1)$. Note that $S_\infty = S_0^{-1}$ and $S_\infty : [-1, 1] \to [0, \infty]$ is an increasing function as are the digit matrices $D_i : [0, \infty] \to [0, \infty]$ $(i = -1, 0, 1)$. Therefore, g_n has the same type (i.e. $+-$ or $-+$) as g. Let D be the first digit of an exact floating point obtained, by the above technique, for y as the unique root of g_n, i.e. $y = S_0 D \cdots$. Then $D_{i_{n+1}} = D$.

This algorithm can also be applied to a function $g : [-1,1] \to [-\infty,+\infty]$ of known type $-+$ or $+-$ with $g(x) \in (-\infty,+\infty)$ for all $x \in (-1,1)$ such that g has a unique root in $(-1,1)$. In this case, however, we cannot compute the signs of $g(1)$ and $g(-1)$ since ∞ has no sign in $\mathbb{R}^*$. But, we can determine in this case whether g is of type $+-$ or $-+$ if there are two distinct points p and q in $(-1,1)$ such that we know

- whether $x \leq p$ or $x \geq p$,
- whether $x \leq q$ or $x \geq q$.

In fact, computing in parallel the signs of $g(p)$ and $g(q)$ reveals the type of g. For example, suppose $p \leq x$ and $q \leq x$. Then, the computation gives us at least one of the two signs and g is of type $+-$ if $g(p) > 0$ or $g(q) > 0$ and of type $-+$ if $g(p) < 0$ or $g(q) < 0$.

This method is efficient to find the unique root of a continuous function g in the interior of a closed rational interval $I \subset [-\infty,+\infty]$ if the graph of g crosses the real line at this root.

A variant of the trisection algorithm can be used to obtain an exact floating point for the unique root x of a continuous $g : (-\infty,+\infty) \to (-\infty,+\infty)$ which is of type $+-$ or $-+$. The algorithm first computes a sign matrix for x; it obtains one of the three sign matrices S_-, S_0 or S_+ representing respectively the intervals $[-\infty,0]$, $[-1,1]$ or $[0,\infty]$. This is achieved by testing the sign of $g(y)$ on the three points $y = -1$, 0 and 1. Then the algorithm proceeds as before to compute the digit matrices.

This algorithm is essentially more general than the iterative algorithm in section 4 as it does not require the canonical extension of the expression tree for f. Furthermore, we have:

Theorem 5.3 *Let* $f : [0,\infty] \to [0,\infty]$ *be a continuous function with* $\bigcap_n f^n[0,\infty] = \{x\}$ *for some* $x \in (0,\infty)$. *Then* $g = f - Id$ *is of type* $+-$ *on* $[0,\infty]$.

Proof

Since any fixed point of f in $[0,\infty]$ is an element of $\bigcap_n f^n[0,\infty] = \{x\}$, it follows that x the unique fixed point of f in $[0,\infty]$. If $f(y) < y$ for some $y < x$ then, it follows from $f(0) > 0$ and the intermediate value theorem that there exists $t \in (0,x)$ with $f(t) = t$ which contradicts the uniqueness of the fixed point x. Hence $f(y) > y$ for all $y < x$. For $y > x$ note that, by assumption $f(\infty)$ is a positive number. Therefore, by continuity of f at ∞, we have $f(K) < K$ for some large real K. Hence, by the intermediate value theorem again, $f(y) < y$ for all $y > x$. $\square$

Therefore the trisection algorithm can also be applied to find the attracting fixed point of a continuous function in the interior of a compact interval.

5.2 Several roots

The above algorithm can be generalized to the case when the map $g : [-1,1] \to (-\infty, +\infty)$ has several roots each of type $+-$ or $-+$. We say that the roots of g are r-isolated for some $r > 0$ if the r-neighbourhoods of the roots (i.e the open balls centred at the roots with radius r) are disjoint. We assume that either we know the number of roots of g in $[-1,1]$ or we know that all roots are r-isolated for some given $r > 0$.

Assume first that we know the number m of roots of g in $[-1,1]$. For each positive integer n, there are $2^{n+1}+1$ dyadic numbers $k/2^n$ with $-2^n \leq k \leq 2^n$ which form an equipartition of $[-1,1]$ with 2^{n+1} subintervals each of length $1/2^n$. We compute the sign of g at the above $2^{n+1}+1$ points in order to locate the m roots. It is sufficient to find $m+1$ points of the partition at which g successively changes sign. Since the computation of the sign of up to m points of this partition may not terminate and at least $m+1$ successful computations are needed, we require $2^{n+1}+1-m \geq m+1$ which gives $2^n \geq m$. Therefore, the algorithm finds the least integer $n \geq \log m$ satisfying the following condition: After obtaining $2^{n+1}+1-m$ successful results in computing in parallel the signs of g at the $2^{n+1}+1$ points of the partition, g successively changes sign m times. There exists some $n \geq \log m$ such that the above condition holds. In fact, suppose l is the minimum distance between the roots and assume $\frac{m+2}{2^n} < l$. Then, between any two successive roots of g, there are at least $m+1$ points of the partition. Therefore, we obtain at least one sign of a point of partition strictly between any two successive roots. Once each root is isolated in the interior of a dyadic interval $[\frac{k}{2^n}, \frac{k'}{2^n}]$ (with $-2^n \leq k < k' \leq 2^n$) at least one sign binary digit (equivalently digit matrix) for it is determined and, then, the basic algorithm is applied to find an exact floating point, equivalently a binary sign expansion, for this isolated root in the dyadic interval.

Assume now that, instead of knowing the number of roots, we know that the roots of g are r-isolated for some given $r > 0$. Let the positive integer n be such that $\frac{1}{2^n} < \frac{r}{2}$. Then for all $-2^n \leq k \leq 2^n - 2$, there is at most one root in $[\frac{k}{2^n}, \frac{k+2}{2^n}]$. Therefore, the algorithm computes the signs of $g(\frac{k}{2^n})$ for $-2^n \leq k \leq 2^n$ in parallel. For each $-2^n \leq k \leq 2^n - 2$, we obtain the sign of g at two elements in $\{\frac{k}{2^n}, \frac{k+1}{2^n}, \frac{k+2}{2^n}\}$. This enables us to isolate all the roots and obtain at least a binary sign digit for each. We then proceed as before to get other digits for each root.

5.3 Implementation

A complete set of algorithms for computing elementary functions in exact floating point has been developed in [15, 14] and implemented in the functional programming language Caml. Using this implementation, the trisection algorithm to find the fixed point and roots of functions in this representation of exact real arithmetic has been implemented in Caml. In practice, in the parallel computation of $g(-\frac{1}{2})$, $g(0)$ and $g(\frac{1}{2})$ the sign is determined, only after 1 or 2 iterations, for all three computations. We give several examples here:

- The function $x \mapsto \frac{x+4}{2x+1}$ has $\sqrt{2}$ as its unique fixed point in $[0, \infty]$. The algorithm gives the exact floating point:

$$S_+ D_1 D_{-1} D_{-1} D_1 D_{-1} D_1 D_{-1} D_1 D_1 D_1 D_1 D_1 D_{-1} \cdots$$

which so far represents the interval $[\frac{4798}{3394}, \frac{4799}{3393}]$.

- The fixed point of $x \mapsto \frac{3x}{4x+1}$ is $\frac{1}{2}$, our algorithm gives

$$S_+ D_{-1} D_1 D_{-1} D_1 D_{-1} D_1 D_{-1} D_1 D_{-1} D_1 D_{-1} D_1 D_{-1} \cdots$$

which so far represents $[\frac{2730}{5462}, \frac{2731}{5461}]$.

- For $x \mapsto \frac{1}{x}$ the result is 1 and we obtain

$$S_+ D_1 D_{-1} D_{-1} D_{-1} D_{-1} D_{-1} D_{-1} D_{-1} D_{-1} D_{-1} \cdots$$

which gives $[1, \frac{513}{512}]$.

- For the fixed point of the function $x \mapsto \arctan(x) + 1$ we get

$$S_+ D_1 D_{-1} D_1 D_0 D_{-1} D_1 D_1 D_{-1} D_{-1} D_0 D_1 \cdots$$

which so far represents $[\frac{5576}{2616}, \frac{5580}{2612}]$.

- For the function $x \mapsto \tan x$ the algorithm gives

$$S_0 \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

which gives zero i.e. the exact result!

6 Concluding remarks

We have presented the iterative and the trisection algorithms for computing fixed points in exact real arithmetic. The former is closely linked with our representation of real numbers using lft's whereas the latter can be applied to any representation of exact real numbers and is generally more efficient as well.

As for future work, we need to investigate the class of the expression trees for which the iterative method converges to the fixed point. The algorithm always converges to a fixed point of the extension of the function to intervals. We have seen that the algorithm does converge to the fixed point of the function itself for any expression tree representing the canonical extension of the function. In fact, convergence to the fixed point of the function takes place for a wider class of expression trees which we need to characterize. Furthermore, we can use the work of Lester, Chambers and Lu (cf [9]) to compute roots of type $--$ or $++$. They use approximations of a real polynomial by rational polynomials to compute approximations of the root of the polynomial. Given an expression tree $\mathcal{T}(x)$ for a function f, the truncated trees $\mathcal{T}_n(x)$, $n \in \mathbb{N}$, give approximations of f by rational fractions with rational coefficients. We can use these truncated trees to compute roots of type $++$ or $--$. Finally, the trisection algorithm can be made more efficient by, for example, using the fact that in practice we often obtain the three signs at the same time.

Acknowledgements

We would like to thank Lindsay Errington, Martin Escardó, Peter Potts and Philipp Sünderhauf for discussions on this work in the Real group at Imperial College. One of the referees brought our attention to [11]. This research has been supported by EPSRC.

References

[1] G. Alefeld and J. Herzberger. *Introduction to Interval Arithmetic.* Academic Press, 1983.

[2] R. Devaney. *An Introduction to Chaotic Dynamical Systems.* Addison Wesley, second edition, 1989.

[3] P. Di Gianantonio. *A functional approach to real number computation.* PhD thesis, University of Pisa, 1993.

[4] P. Di Gianantonio. An abstract data type for real numbers. In *Proceedings of 24th International Colloquium on Automata, Languages, and Programming (ICALP'97)*, 1997.

[5] A. Edalat and P. J. Potts. A new representation for exact real numbers. In *Proceedings of Mathematical Foundations of Programming Semantics 13*, volume 6 of *Electronic Notes in Theoretical Computer Science.* Elsevier Science B. V., 1997. Available from URL: http://www.elsevier.nl/locate/entcs/volume6.html.

[6] A. Edalat, P.J. Potts, and P. Sünderhauf. Lazy computation with exact real numbers. In *Proceedings of ICFP98*, 1998. Available on WWW in `http://theory.doc.ic.ac.uk/~ae`.

[7] M. H. Escardó. PCF extended with real numbers. *Theoretical Computer Science*, 162(1):79–115, August 1996.

[8] W. Gosper. *Continued Fraction Arithmetic.* HAKMEM Item 101B, MIT Artificial Intelligence Memo 239. MIT, 1972.

[9] D. Lester, S. Chambers, and H. L. Hu. The computable eigenvalue problem. Proceedings of the Third Real Numbers and Computers Conference, 1998.

[10] D. R. Lester. Vuillemin's exact real arithmetic. In R. Heldal, C. K. Holst, and P. L. Wadler, editors, *Functional Programming, Glasgow 1991: Proceedings of the 1991 Workshop, Portree, UK*, pages 225–238, Berlin, 1992. Springer-Verlag.

[11] H. Luckhardt. A fundamental effect in computations on real numbers. *Theoretical Computer Science*, 5:321–324, 1977.

[12] A. Nielsen and P. Kornerup. Msb-first digit serial arithmetic. *J. of Univ. Comp. Scien.*, 1(7):523–543, 1995.

[13] P. Potts, A. Edalat, and M. Escardó. Semantics of exact real arithmetic. In *Twelfth Annual IEEE Symposium on Logic in Computer Science*. IEEE, 1997.

[14] P. J. Potts. Efficient on-line computation of real functions using exact floating point. Submitted for publication. Available from: `http://theory.doc.ic.ac.uk/~pjp`, 1997.

[15] P. J. Potts and A. Edalat. Exact Real Computer Arithmetic, March 1997. Department of Computing Technical Report DOC 97/9, Imperial College, available from `http://theory.doc.ic.ac.uk/~ae`.

[16] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C.* CUP, 1992.

[17] J. E. Vuillemin. Exact real computer arithmetic with continued fractions. *IEEE Transactions on Computers*, 39(8):1087–1105, 1990.

[18] K. Weihrauch. *Computability*, volume 9 of *EATCS Monographs on Theoretical Computaer Science*. Springer-Verlag, 1987.

[19] E. Wiedmer. Computing with infinite objects. *Theoretical Computer Science*, 10:133–155, 1980.