An exercise on streams: convergence acceleration (short version)

Pierre Lescanne
University of Lyon,
École normale supérieure de Lyon,
LIP (UMR 5668 CNRS ENS Lyon UCBL INRIA)
46 allée d'Italie, 69364 Lyon, France

1 Introduction

Assume that we want to compute numerically the limit of a sequence that converges slowly. If we use the sequence itself, we will get significant figures of the limit after a long time. Methods called *convergence acceleration* have been designed to shorten the time after which we get reasonable amount of significant digits. In other words, *convergence acceleration* is a set of methods for numerically computing the limit of a sequence of numbers. Those methods are based on sequence transformations and are a nice domain of application of streams, with beautiful higher order functions. This allows us to present elegantly rather complex methods and to code them in Haskell replacing long, obscure and special purpose Fortran programs by short, generic and arbitrary precision Haskell programs.

In this paper we show, how given a sequence $(s_n)_{n\in\mathbb{N}}$, we can evaluate efficiently $\lim_{n\to\infty} s_n$. For that we use *Levin transforms*. There are three kinds of such transforms, which are the result of three sequence transformations labeled traditionally by t, u and v.

A longer paper associated with this work is:

Pierre Lescanne An exercise on streams: convergence acceleration, arXiv:1312.4917 `http://arxiv.org/abs/1312.4917`

2 Presentation of the method

We use Haskell and we work with arbitrary precision reals based on the implementation of David Lester, called *CReal*. In Haskell the type of streams over a type A is written $[A]$.

For the numerical aspect, we follows Naoki Osada and we show that the stream notation of Haskell makes the presentation simpler. With no surprise, the size of the Haskell code is the same if not shorter than the mathematical description of the reference. Moreover it provides efficient programs.

Levin transformations are somewhat generic in the sense that they are based on elementary transformations. Specialists of convergence acceleration propose three such elementary transformations. Let s be a sequence on *CReal*, i.e., $s :: [CReal]$. We define first a basic sequence transformation on which we will found our elementary transformations:

$$dELTA :: [CReal] \to [CReal]$$
$$dELTA\ s = zipWith\ (-)\ (tail\ s)\ s$$

which means that $dELTA(s)_n = s_{n+1} - s_n$. From this basic sequence transformation we define the three elementary other sequence transformations as follows. A unique function depending on a character parameter which is either $'t'$ or $'u'$ or $'v'$ is given. It corresponds to the traditional notations of numerical analysis for those sequence transformations:

```
delta :: Char → [CReal] → [CReal]
delta 't' s = dELTA s
delta 'u' s = zipWith (*) (dELTA s) [1..]
delta 'v' s = zipWith (/) (zipWith (*) (tail$dELTA s) (dELTA s))
                          (dELTA (dELTA s))
```

In numerical analysis, people speak about *E-algorithm*. This is a family of functions $eAlg_{n,k}$ which are also parametrized by a character either $'t'$ or $'u'$ or $'v'$. It tells which of the basic sequence transformations is chosen. $eAlg_{n,k}$ uses a family of auxiliary functions which we call $gAlg_{n,k}$ for symmetry and regularity. Here is the Haskell code for these functions:

```
eAlg :: Char → Int → [CReal] → [CReal]
eAlg c 0 s = s
eAlg c k s = let
               a = (eAlg c (k-1) s)
               b = (gAlg c (k-1) k s)
             in
             zipWith (-) a (zipWith (*) b (zipWith (/) (dELTA a) (dELTA b)))
```

```
gAlg :: Char → Int → Int → [CReal] → [CReal]
gAlg c 0 j s = let
                 nTojMinus1 j = zipWith (**) [1..] (map fromIntegral [j-1, j-1..])
               in
               zipWith (/) (nTojMinus1 j) (delta c s)
gAlg c k j s = let
                 a = gAlg c (k-1) j s
                 b = gAlg c (k-1) k s
               in
               zipWith (-) a (zipWith (*) b (zipWith (/) (dELTA a) (dELTA b)))
```

Here is the formula as it is given in Osada. R_n is the generic value of $(delta\ c\ s)_n$. $gAlg$ is written g. $E_k^{(n)}$ is the n^{th} element of the sequence $eAlg\ c\ k\ s$, the same for $g_{k,j}^{(n)}$. Δ is the notation for what we write $dELTA$.

$$
\begin{aligned}
E_0^{(n)} &= s_n, \quad g_{0,j}^{(n)} = n^{1-j} R_n \quad , && n = 1, 2, ...;\ j = 1, 2, ..., \\
E_k^{(n)} &= E_{k-1}^{(n)} - g_{k-1,k}^{(n)} \frac{\Delta E_{k-1}^{(n)}}{\Delta g_{k-1,k}^{(n)}} \ , && n = 1, 2, ...;\ k = 1, 2, ..., \\
g_{k,j}^{(n)} &= g_{k-1,j}^{(n)} - g_{k-1,k}^{(n)} \frac{\Delta g_{k-1,j}^{(n)}}{\Delta g_{k-1,k}^{(n)}}, && n = 1, 2, ...;\ k = 1, 2, ..., j > k
\end{aligned}
$$