

Accurate simple zeros of polynomials in floating point arithmetic

Stef Graillat*

November 8, 2007

Abstract

In the paper, we examine the behavior of the Newton's method in floating point arithmetic for the computation of a simple zero of a polynomial. We allow an extended precision (twice the working precision) in the computation of the residual. We prove that, for a sufficient number of iteration, the zero is as accurate as if computed in twice the working precision. We provides numerical experiments confirming this.

Key words: zeros of polynomials, Newton's method, condition number, floating point arithmetic

AMS Subject Classifications: 12D10, 30C10, 30C15, 26C10

1 Introduction and notation

The key to compute an accurate solution to a nonlinear equation is the accurate evaluation of the function in use. In this paper, our purpose is to compute accurate simple zeros of univariate polynomials relying on Newton's method. To reach this aim, we need to focus on two important things:

- explaining what we mean by “accurate solution”;
- having an accurate polynomial evaluation algorithm to compute the residual in the Newton's iteration.

Let us explain now what we mean by “accurate solution”. Let \hat{x} be the computed solution of a problem (P) whose exact solution is x . Suppose that the computations have been done with a t -bit floating point arithmetic. We will say the \hat{x} is as accurate as if computed with twice the working precision if

$$\frac{|\hat{x} - x|}{|x|} \leq \text{eps} + C\text{eps}^2 \text{cond}(P). \quad (1.1)$$

where C is a moderate constant, $\text{eps} = 2^{-t}$, $|\cdot|$ is a norm on the space of the solution and $\text{cond}(P)$ is the condition number of the problem (P). In the right-hand side of inequality (1.1), the second term reflects the computation in twice the working precision and the first one the

*Laboratoire LIP6, Département Calcul Scientifique, Université Pierre et Marie Curie (Paris 6), 4 place Jussieu, F-75252, Paris cedex 05, France (stef.graillat@lip6.fr, <http://www-pequan.lip6.fr/~graillat>).

rounding into the working precision. Relation (1.1) is what we called the compensated rule of thumb, the classic rule of thumb being [5, p.9]

$$\frac{|\widehat{x} - x|}{|x|} \leq C \text{eps} \text{cond}(P).$$

Throughout the paper, we assume to work with a floating point arithmetic adhering to IEEE 754 floating point standard [6]. We assume that neither overflow nor underflow occur. The set of floating point numbers is denoted by \mathbf{F} and the relative rounding error by eps . For IEEE 754 double precision we have $\text{eps} = 2^{-53}$ and for IEEE 754 single precision $\text{eps} = 2^{-24}$.

We denote by $\text{fl}(\cdot)$ the result of a floating point computation, where all operations inside parentheses are done in floating point working precision. Floating point operations in IEEE 754 satisfy [5]

$$\text{fl}(a \circ b) = (a \circ b)(1 + \varepsilon) = \text{ for } \circ = \{+, -, \cdot, /\} \text{ and } |\varepsilon| \leq \text{eps}.$$

This implies that

$$|a \circ b - \text{fl}(a \circ b)| \leq \text{eps}|a \circ b| \text{ and } |a \circ b - \text{fl}(a \circ b)| \leq \text{eps}|\text{fl}(a \circ b)| \text{ for } \circ = \{+, -, \cdot, /\}.$$

We use standard notation for error estimations. The quantities γ_n are defined as usual [5] by

$$\gamma_n := \frac{n \text{eps}}{1 - n \text{eps}} \text{ for } n \in \mathbf{N},$$

where we implicitly assume that $n \text{eps} \leq 1$.

The rest of the paper is organized as follows. In Section 2, we recall some results on Horner scheme, error-free transformations and the Compensated Horner scheme. In Section 3, we present the condition number of a simple zero. In Section 4, we present the Newton's method for root-finding using the Compensated Horner scheme to compute the residual. In Section 5, we give some numerical experiments. Finally, we conclude by giving some hints about future work.

2 Accurate polynomial evaluation

In this section, we first recall the Horner scheme as well as give an error bound. We then recall the classic error-free transformations. We use these transformations for a Compensated Horner scheme which gives a result as accurate as if computed by the classic Horner scheme using twice the working precision and then rounded to the working precision.

2.1 Classic Horner Scheme

The classic method for evaluating a polynomial

$$p(x) = \sum_{i=0}^n a_i x^i$$

is the Horner scheme which consists in the following algorithm.

Algorithm 2.1. Polynomial evaluation with Horner's scheme

function `res = Horner(p, x)`

```

 $s_n = a_n$ 
for  $i = n - 1 : -1 : 0$ 
     $s_i = s_{i+1} \cdot x + a_i$ 
end
res = s_0
```

A forward error bound is (see [5, p.95]):

$$|p(x) - \text{Horner}(p, x)| \leq \gamma_{2n} \sum_{i=0}^n |a_i| |x|^i = \gamma_{2n} \tilde{p}(|x|) \quad (2.2)$$

where $\tilde{p}(x) = \sum_{i=0}^n |a_i| x^i$.

2.2 Error-free transformations (EFT)

One can notice that $a \circ b \in \mathbf{R}$ and $\text{fl}(a \circ b) \in \mathbf{F}$ but in general we do not have $a \circ b \in \mathbf{F}$. It is known that for the basic operations $+$, $-$, \cdot , the approximation error of a floating point operation is still a floating point number (see for example [3]):

$$\begin{aligned} x = \text{fl}(a \pm b) &\Rightarrow a \pm b = x + y \quad \text{with } y \in \mathbf{F}, \\ x = \text{fl}(a \cdot b) &\Rightarrow a \cdot b = x + y \quad \text{with } y \in \mathbf{F}, \end{aligned} \quad (2.3)$$

where no underflow is assumed for multiplication. These are *error-free* transformations of the pair (a, b) into the pair (x, y) .

Fortunately, the quantities x and y in (2.3) can be computed exactly in floating point arithmetic. For the algorithms, we use Matlab-like notations. For addition, we can use the following algorithm by Knuth [7, Thm B. p.236].

Algorithm 2.2 (Knuth [7]). Error-free transformation of the sum of two floating point numbers

```
function [x, y] = TwoSum(a, b)
    x = fl(a + b)
    z = fl(x - a)
    y = fl((a - (x - z)) + (b - z))
```

For the error-free transformation of a product, we first need to split the input argument into two parts. Let p be given by $\mathbf{eps} = 2^{-p}$ and define $s = \lceil p/2 \rceil$. For example, if the working precision is IEEE 754 double precision, then $p = 53$ and $s = 27$. The following algorithm by Dekker [3] splits a floating point number $a \in \mathbf{F}$ into two parts x and y such that

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ nonoverlapping with } |y| \leq |x|.$$

Algorithm 2.3 (Dekker [3]). Error-free split of a floating point number into two part

```
function [x, y] = Split(a, b)
    factor = fl(2^s + 1)
    c = fl(factor * a)
    x = fl(c - (c - a))
    y = fl(a - x)
```

With this function, an algorithm from Veltkamp (see [3]) makes it possible to compute an error-free transformation for the product of two floating point numbers. This algorithm returns two floating point numbers x and y such that

$$a \cdot b = x + y \quad \text{with } x = \text{fl}(a \cdot b).$$

Algorithm 2.4 (Veltkamp [3]). Error-free transformation of the product of two floating point numbers

```

function  $[x, y] = \text{TwoProduct}(a, b)$ 
   $x = \text{fl}(a \cdot b)$ 
   $[a_1, a_2] = \text{Split}(a)$ 
   $[b_1, b_2] = \text{Split}(b)$ 
   $y = \text{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$ 

```

The following theorem summarizes the properties of algorithms `TwoSum` and `TwoProduct`.

Theorem 2.1 (Ogita, Rump and Oishi [8]). *Let $a, b \in \mathbf{F}$ and let $x, y \in \mathbf{F}$ such that $[x, y] = \text{TwoSum}(a, b)$ (Algorithm 2.2). Then,*

$$a + b = x + y, \quad x = \text{fl}(a + b), \quad |y| \leq \text{eps}|x|, \quad |y| \leq \text{eps}|a + b|. \quad (2.4)$$

The algorithm `TwoSum` requires 6 flops.

Let $a, b \in \mathbf{F}$ and let $x, y \in \mathbf{F}$ such that $[x, y] = \text{TwoProduct}(a, b)$ (Algorithm 2.4). Then,

$$a \cdot b = x + y, \quad x = \text{fl}(a \cdot b), \quad |y| \leq \text{eps}|x|, \quad |y| \leq \text{eps}|a \cdot b|. \quad (2.5)$$

The algorithm `TwoProduct` requires 17 flops.

We present now an error-free transformation for the polynomial evaluation with Horner scheme.

Algorithm 2.5 (Graillat, Langlois and Louvet [4]). Error-free transformation for the Horner scheme

```

function  $[h, p_\pi, p_\sigma] = \text{EFTHorner}(p, x)$ 
   $s_n = a_n$ 
  for  $i = n - 1 : -1 : 0$ 
     $[p_i, \pi_i] = \text{TwoProduct}(s_{i+1}, x)$ 
     $[s_i, \sigma_i] = \text{TwoSum}(p_i, a_i)$ 
    Let  $\pi_i$  be the coefficient of degree  $i$  in  $p_\pi$ 
    Let  $\sigma_i$  be the coefficient of degree  $i$  in  $p_\sigma$ 
  end
   $h = s_0$ 

```

The next theorem proves that Algorithm 2.5 is an error-free transformation.

Theorem 2.2 (Graillat, Langlois and Louvet [4]). *Let $p(x) = \sum_{i=0}^n a_i x^i$ be a polynomial of degree n with floating point coefficients, and let x be a floating point value. Let $[h, p_\pi, p_\sigma] = \text{EFTHorner}(p, x)$ (Algorithm 2.5). Then*

- i) the floating point evaluation $h = \text{Horner}(p, x)$ and*
- ii) two polynomials p_π and p_σ of degree $n - 1$ with floating point coefficients,*

satisfies

$$p(x) = h + (p_\pi + p_\sigma)(x). \quad (2.6)$$

Algorithm 2.5 requires $23n$ flops.

2.3 Compensated Horner Scheme

From Theorem 2.2, the global forward error affecting the floating point evaluation of p at x according to the Horner scheme is

$$e(x) = p(x) - \text{Horner}(p, x) = (p_\pi + p_\sigma)(x).$$

The coefficients of these polynomials are exactly computed by Algorithm 2.5, together with $\text{Horner}(p, x)$. Indeed, if $[h, p_\pi, p_\sigma] = \text{EFTHorner}(p, x)$, then p_π and p_σ are two exactly representable polynomials. The key to increase the accuracy of the computed result is to compute an approximate of the global error $e(x)$ in working precision, and then to compute a corrected result

$$\mathbf{res} = \text{fl}(\text{Horner}(p, x) + e(x)).$$

We say that $c = \text{fl}(e(x))$ is a corrective term for $\text{Horner}(p, x)$. The corrected result \mathbf{res} is expected to be more accurate than the first result $\text{Horner}(p, x)$.

Our aim is now to compute the corrective term $c = \text{fl}((p_\pi + p_\sigma)(x))$. For that we evaluate the polynomial whose coefficients are those of $p_\pi + p_\sigma$ rounded to the nearest floating point value. This process is described by Algorithm 2.6.

Algorithm 2.6. Evaluation of the sum of two polynomials.

```
function res = HornerSum(p, q, x)
    r_n = fl(a_n + b_n)
    for i = n - 1 : -1 : 0
        r_i = fl(r_{i+1} · x + (a_i + b_i))
    end
```

We can now describe the Compensated Horner Scheme.

Algorithm 2.7 (Graillat, Langlois and Louvet [4]). Compensated Horner scheme

```
function res = CompHorner(p, x)
    [h, p_pi, p_sigma] = EFTHorner(p, x)
    c = HornerSum(p_pi, p_sigma, x)
    res = fl(h + c)
```

The following theorem proves that the result of a polynomial evaluation computed with the Compensated Horner scheme (2.7) is as accurate as if computed by the classic Horner scheme using twice the working precision and then rounded to the working precision.

Theorem 2.3 (Graillat, Langlois and Louvet [4]). *Given a polynomial $p = \sum_{i=0}^n a_i x^i$ of degree n with floating point coefficients, and x a floating point value. We consider the result $\text{CompHorner}(p, x)$ computed by Algorithm 2.7. Then,*

$$|\text{CompHorner}(p, x) - p(x)| \leq \text{eps}|p(x)| + \gamma_{2n}^2 \tilde{p}(x). \quad (2.7)$$

The Algorithm CompHorner requires $26n + 3$ flops.

3 Condition number for root finding

Given a problem, we want to know how to measure the difficulty of solving it. This will be done via the notion of *condition number*. Roughly speaking, the condition number measures the sensitivity of the solution to perturbation in the data. Here is the classic definition for the condition number of root finding for simple roots.

Definition 3.1. Let $p(z) = \sum_{i=0}^n a_i z^i$ be a polynomial of degree n and x be a simple zero of p . The *condition number* of x is defined by

$$\text{cond}(p, x) = \limsup_{\varepsilon \rightarrow 0} \left\{ \frac{|\Delta x|}{\varepsilon |x|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}.$$

In the previous definition, Δx represents the variation of the zero x when the polynomial is perturbed by a polynomial $\Delta p(z) = \sum_{i=0}^n \Delta a_i z^i$. It means that $x + \Delta x$ is a zero of $p + \Delta p$.

The following theorem gives an explicit formula to compute the condition number.

Theorem 3.1 (Chaitin-Chatelin and Frayssé [2]). Let p be a polynomial of degree n and x be a simple zero of p . The condition number of x is given by

$$\text{cond}(p, x) = \frac{\tilde{p}(|x|)}{|x| |p'(x)|}.$$

Proof. We recall the proof done in [2]. Let $p(z) = \sum_{i=0}^n a_i z^i$ and consider the map

$$\varphi : \begin{cases} \mathbf{C}^{n+1} & \longrightarrow \mathbf{C}, \\ (a_0, \dots, a_n)^T & \longmapsto z \text{ such that } p(z) = 0, z \text{ simple.} \end{cases}$$

From the definition of φ , it follows that $p(\varphi(p)) = 0$. The chain rule gives

$$\frac{\partial \varphi}{\partial p_i}(p) p'(z) + z^i = 0, \quad i = 0 : n.$$

This implies that $\varphi'(p) = -\frac{1}{p'(z)} \underline{z}^T$. Using Taylor expansion, we obtain

$$x + \Delta x = x - \frac{1}{p'(x)} \underline{x}^T \Delta p + \mathcal{O}(\|\Delta p\|^2). \quad (3.8)$$

From Definition 3.1, it follows that

$$\text{cond}(p, x) = \frac{1}{|x| |p'(x)|} \sup \{ |\underline{x}^T \Delta p|, |\Delta a_i| \leq |a_i| \}.$$

It is clear now that

$$\text{cond}(p, x) = \frac{|\underline{x}^T p|}{|x| |p'(x)|} = \frac{\tilde{p}(|x|)}{|x| |p'(x)|}.$$

□

4 Accurate Newton's method

In this section, we present the analysis of [9] for the Newton's method in floating point arithmetic (see also [1]). We specialize the result from [9] in the case of an univariate polynomial with a simple root. We use the Compensated Horner scheme to accurately compute the residual. In that case, we show that the computed result (an approximation of the simple root of the polynomial) is as accurate as if computed with twice the working precision via the classic Newton's method and then rounded back to the working precision.

4.1 General results on Newton's method

In [9], F. Tisseur provided a comprehensive analysis of the Newton's method in floating point arithmetic. We recall hereafter her analysis and her results (see also [5, chap.25]).

Let $F : \mathbf{R}^m \rightarrow \mathbf{R}^m$ be continuously differentiable on \mathbf{R}^m and J the Jacobian matrix $(\partial F_i / \partial x_j)$ of F . We assume that J is Lipschitz continuous with constant β in \mathbf{R}^m , that is,

$$\|J(w) - J(v)\| \leq \beta \|w - v\| \text{ for all } v, w \in \mathbf{R}^m,$$

where $\|\cdot\|$ denotes any vector norm and the corresponding operator norm. We denote by $\kappa(J) = \|J\| \|J^{-1}\|$ the condition number of the matrix J . We aim at solving the system of nonlinear equations $F(x) = 0$ by Newton's method:

$$J(x_i)(x_{i+1} - x_i) = -F(x_i), \quad i \geq 0, \quad (4.9)$$

for a given point x_0 . Equation (4.9) can be rewritten as

$$\begin{aligned} \text{Solve } J(x_i)d_i &= -F(x_i), \\ x_{i+1} &= x_i + d_i. \end{aligned}$$

Due to rounding errors in floating point arithmetic, we have

$$\widehat{x}_{i+1} = \widehat{x}_i - (J(\widehat{x}_i) + E_i)^{-1}(F(\widehat{x}_i) + e_i) + \varepsilon_i,$$

where

- e_i is the error made during the computation of the residual $F(x_i)$,
- E_i is the error when forming $J(\widehat{x}_i)$ and solving the linear system for d_i ,
- ε_i is the error made when adding \widehat{d}_i to \widehat{x}_i .

We assume that $F(\widehat{x}_i)$ is computed with extended precision \mathbf{eps}' and then rounded back to working precision \mathbf{eps} , and that $\widehat{d}_i, \widehat{x}_i$ are computed at precision \mathbf{eps} . Hence we assume that there exists a function ψ depending on F, x_i, \mathbf{eps} and \mathbf{eps}' such that

$$\|e_i\| \leq \mathbf{eps} \|F(\widehat{x}_i)\| + \psi(F, \widehat{x}_i, \mathbf{eps}, \mathbf{eps}').$$

We assume that the error E_i satisfies

$$\|E_i\| \leq \mathbf{eps} \varphi(F, \widehat{x}_i, n, \mathbf{eps}).$$

For the error ε_i , we have $\|\varepsilon_i\| \leq \mathbf{eps} (\|\widehat{x}_i\| + \|\widehat{d}_i\|)$.

Theorem 4.1 (Tisseur [9, Cor. 2.3]). *Assume that there is an x such that $F(x) = 0$ and $J = J(x)$ is nonsingular and satisfies*

$$\mathbf{eps} \kappa(J) \leq 1/8$$

where $\kappa(J) = \|J\| \|J^{-1}\|$ denotes the condition number of the matrix J . Assume also that for φ ,

$$\mathbf{eps} \|J(\widehat{x}_i)^{-1}\| \varphi(F, \widehat{x}_i, n, \mathbf{eps}) \leq 1/8 \text{ for all } i.$$

Then, for all x_0 such that

$$\beta \|J^{-1}\| \|x_0 - x\| \leq 1/8,$$

Newton's method in floating point arithmetic generates a sequence of $\{\widehat{x}_i\}$ whose normwise relative error decreases until the first i for which

$$\frac{\|\widehat{x}_{i+1} - x\|}{\|x\|} \approx \mathbf{eps} + \frac{\|J^{-1}\|}{\|x\|} \psi(F, \widehat{x}_i, \mathbf{eps}, \mathbf{eps}'). \quad (4.10)$$

4.2 Newton's method for polynomials

Let p be a given polynomial with simple zeros. We apply the Newton's method with $F(x) = p(x)$ and so $J(x) = p'(x)$. The classic Newton's method is Algorithm 4.1.

Algorithm 4.1. Classic Newton's method

$$\begin{aligned} x_0 &= \xi \\ x_{i+1} &= x_i - \frac{p(x_i)}{p'(x_i)} \end{aligned}$$

Hereafter, we use the compensated Horner scheme to evaluate the residual $p(x)$ in order to get a result as accurate as if computed in twice the working precision. We also assume that we already know that the root we are looking for belongs to $[a, b]$ with $a, b \in \mathbf{R}$. We also define $\beta = \max_{x \in [a, b]} |p'(x)|$. The accurate Newton's method is Algorithm 4.2.

Algorithm 4.2. Accurate Newton's method

$$\begin{aligned} x_0 &= \xi \\ x_{i+1} &= x_i - \frac{\text{CompHorner}(p, x_i)}{p'(x_i)} \end{aligned}$$

In that case, using notation of

4.3

sub:newton, we have $\text{eps}' = \text{eps}^2$ and $\psi(F, \hat{x}_i, \text{eps}, \text{eps}') = \gamma_{2n}^2 \tilde{p}(x)$ thanks to Theorem 2.3. Moreover, using Theorem 4.1 and Theorem 3.1, it holds the following theorem.

Theorem 4.2. *Assume that there is an x such that $p(x) = 0$ and $p'(x) \neq 0$ is not too small. Assume also that*

$$\text{eps} \text{cond}(p, x) \leq 1/8 \text{ for all } i.$$

Then, for all x_0 such that

$$\beta |p'(x)|^{-1} |x_0 - x| \leq 1/8,$$

Newton's method in floating point arithmetic generates a sequence of $\{x_i\}$ whose normwise relative error decreases until the first i for which

$$\frac{|x_{i+1} - x|}{|x|} \approx \text{eps} + \gamma_{2n}^2 \text{cond}(p, x). \quad (4.11)$$

The theorem means that if we begin not too far from the simple root, the Newton's method gives an approximation of the root as accurate as if computed with twice the working precision.

The use of an accurate polynomial evaluation algorithm is essential. Indeed, if we use the classic Horner scheme, then, at the end of the iteration, we only have

$$\frac{|x_{i+1} - x|}{|x|} \approx \gamma_{2n} \text{cond}(p, x). \quad (4.12)$$

5 Numerical experiments

All our experiments are performed using the IEEE-754 double precision with MATLAB 7. When needed, we use the Symbolic Math Toolbox to accurately compute the roots of polynomials (in order to compute the relative forward error).

We test the Newton's iterations on the expanded form of the polynomial $p_n(x) = (x-1)^n - 10^{-8}$ for $n = 1 : 40$. The condition number $\text{cond}(p_n, x)$ where x is the root $1 + 10^{-8/n}$ varies from 10^4 to 10^{22} .

Figure 1 shows the relative accuracy $|\hat{x} - x|/|x|$ where x the exact root and \hat{x} is the computed value by the two algorithms 4.1 and 4.2. We also plot the *a priori* error estimation (4.12) and (4.11).

As we can see in Figure 1, the accurate Newton's iteration exhibit the expected behavior, that is to say, the compensated rule of thumb. As long as the condition number is less than 10^{15} , the accurate Newton's iteration produce results with full precision (forward relative error of the order of 10^{-16}). For condition numbers greater than 10^{15} , the accuracy decreases.

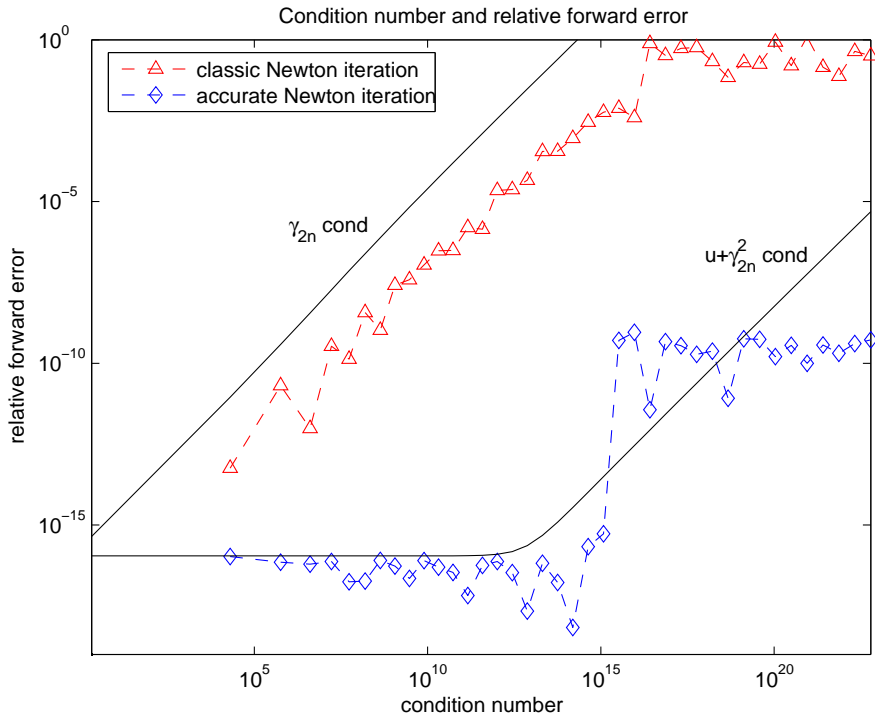


Figure 1: Accuracy of the classic Newton iteration and the accurate Newton iteration

6 Conclusion and future work

In the paper, we have proved that the Newton's iteration makes it possible to compute a good approximation of a simple root; by good approximation, we mean as accurate as if computed with twice the working precision.

We only dealt with simple roots. If the root has multiplicity $m > 1$, one can use the modified Newton's iteration as follows.

Algorithm 6.1. Modified Newton's method

$$x_0 = \xi$$

$$x_{i+1} = x_i - m \frac{p(x_i)}{p'(x_i)}$$

A future work will be to see if we can get the same kind of results than for simple roots when we already know the multiplicity of the root.

References

- [1] Daniel J. Bates, Andrew J. Sommese, Jonathan D. Hauenstein, and Charles W. Wampler. Adaptive multiprecision path tracking. *SIAM J. Numer. Anal.*, 2007. To appear.
- [2] Françoise Chaitin-Chatelin and Valérie Frayssé. *Lectures on finite precision computations*. Software, Environments, and Tools. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.
- [3] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.
- [4] Stef Graillat, Nicolas Louvet, and Philippe Langlois. Compensated Horner scheme. Research Report 04, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, July 2005.
- [5] Nicholas J. Higham. *Accuracy and stability of numerical algorithms*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2002.
- [6] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985*. Institute of Electrical and Electronics Engineers, New York, 1985. Reprinted in SIGPLAN Notices, 22(2):9–25, 1987.
- [7] Donald E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. Addison-Wesley, Reading, MA, USA, third edition, 1998.
- [8] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.
- [9] Françoise Tisseur. Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems. *SIAM J. Matrix Anal. Appl.*, 22(4):1038–1057 (electronic), 2001.