INTEGER ROOTS FOR INTEGER-POWER-CONTENT CALCULATIONS

D.J. JEFFREY, M.W. GIESBRECHT AND R.M. CORLESS
The University of Western Ontario, London, Ontario, Canada
E-mail: D.Jeffrey,M.Giesbrecht,R.Corless@uwo.ca

In this extended abstract, we outline investigations of the application of Newton and Halley iteration to the computation of nth roots of an integer. We give an analysis that reduces the number of iterations by guaranteeing the number of correct digits obtained at each iteration. The initial application is to the calculation of the integer-power content of an integer.

1 Introduction

The *integer-power content* of $a \in \mathbb{Z}$ is defined[1] to be the largest $n \in \mathbb{Z}$ such that $a = b^n$ for some $b \in \mathbb{Z}$. We write $\mathrm{ipc}(a) = n$. Also, $b = \mathrm{ipf}(a)$ is the *integer-power free* base for a. Clearly we have $a = \mathrm{ipf}(a)^{\mathrm{ipc}(a)}$. Any algorithm that finds the integer-power content and base is called a perfect-power classification algorithm by Bernstein[2]. The first step in such an algorithm will be what Bernstein calls a perfect-power decomposition algorithm, that is the finding of any integer n, not necessarily maximal, such that $a = b^n$. Algorithms for computing n have been given by Bach & Sorenson[3], by Bernstein[2], and others. The computer algebra program Maple has a function `iperfpow` which computes a perfect power. In Maple 6, this function does not compute the integer-power content, as can be illustrated by the maple session

```
> iperfpow(256);
                              16
```

A critical part of all the algorithms is the computation of the integer root of an integer, together with a test of whether this is an exact root. The Maple command `iroot(a,n)` will compute the integer closest to $a^{1/n}$, for $a, n \in \mathbb{Z}$, but the command does not offer any test for an exact root.

Most algorithms proposed for root extraction use iterative schemes based on Newton iteration. One exception is the algorithm used by the `iroot` function in Maple release 6. The function `iroot(a,n)` rounds the result of the floating-point computation of $\exp\left(\frac{1}{n}\ln a\right)$. The algorithm of Bach and Sorensen uses a novel form of Newton iteration based on taking the floor of all intermediate results. We show that although this algorithm is attractive, it is not well suited to the Maple environment, and in general will become less efficient as the integers in question become larger. The algorithm of Bernstein

is based on a progressive-precision Newton iteration. The use of progressively increasing precision at each step of an iteration is something that is obvious and instinctive to human computers working by hand, and was enunciated in a computer setting by Brent[4].

As well as examining the efficiency of existing algorithms, including p-adic methods, this paper considers in detail the control loops used in programming progressive-precision iteration. We also consider Halley iteration as an alternative to Newton iteration. Our considerations are also influenced by the specifics of the Maple computing environment, in particular the fact that Maple offers a radix 10 number system.

2 Newton and Halley Iteration

In this section, we give a uniform treatment of the Newton and Halley iterations.

2.1 General iteration formulae

Consider solving the equation $f(x) = 0$, given an initial estimate x_0 for the solution. We expand $f(x)$ as a Taylor series around x_0.

$$f(x) = f(x_0)+(x-x_0)f'(x_0)+\tfrac{1}{2}(x-x_0)^2f''(x_0)+\tfrac{1}{6}(x-x_0)^3f'''(x_0)+\dots \quad (1)$$

Setting $h = x-x_0$ and $f(x) = 0$, we can solve for h by the Lagrange inversion theorem. Abbreviating $f(x_0)$ to f for clarity, we write

$$h = -\frac{1}{f'}f - \frac{f''}{2(f')^3}f^2 + \frac{3(f'')^2 - f'f'''}{6(f')^5}f^3 + \dots \quad (2)$$

The series is written as shown to emphasize that it is a series in powers of $f(x_0)$. The classical Newton iteration is obtained by taking one term of this series, and the classical Halley iteration is obtained by converting the series to a continued fraction and taking terms to $O(f^2)$. The continued fraction form of (2) is

$$h = \cfrac{-f}{f' + \cfrac{-f}{2f'/f'' + \cfrac{-f}{3f'(f'')^2/(3(f'')^2 - 2f'f''')}}}\,, \quad (3)$$

and dropping higher-order terms and reverting to standard iteration notation, we get Halley's method as

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k) - \frac{1}{2}f(x_k)f''(x_k)/f'(x_k)}\,. \quad (4)$$

2.2 Rate of convergence

The accuracy of the iteration can be analyzed in terms of forward and backward error. If the exact solution to $f = 0$ is denoted x_e, then an estimate x_0 gives a forward (absolute) error of $|x_e - x_0|$ and a backward error of $|f(x_0)|$. Maple users are accustomed to specifying accuracy by setting '`Digits`', in other words specifying the number of correct digits in the answer. This is essentially the forward error. The present root-finding problem also requires a guaranteed forward error. The popular rules of thumb are that Newton 'doubles the number of correct digits each iteration', and Halley 'triples the number of correct digits'; the theorems below show that these rules are approximations.

Theorem. With the above notation, let x_0 be a d-digit approximation in radix B to x_e. So $|x_e - x_0|/|x_e| = bB^{-d+1}$, where $0 < b < 1$. Then one step of a Newton iteration gives $x_1^{(n)}$ accurate to

$$\frac{|x_1^{(n)} - x_e|}{|x_e|} = \frac{|b^2 B^{-2d+2} x_0 f''(\xi)|}{|2f'|} \ , \tag{5}$$

for some $x_0 < \xi < x_e$. The improvement in one step of a Halley method is

$$\frac{|x_1^{(h)} - x_e|}{|x_e|} = \frac{|(x_e - x_0)^3 f'''(\xi)|}{|6x_0 f'(x_0)|} = \frac{|b^3 B^{-3d+3} x_0^2 f'''(\xi)|}{|6f'(x_0)|} \tag{6}$$

3 Algorithms for root extraction

One standard algorithm for extracting a root is the p-adic algorithm[5]. This has not been used by any of the other papers studying perfect powers, the reason being that a p-adic iteration returns an integer, even if the root being calculated is not exact. A separate test must then be made to detect this.

An integer-based algorithm has been given by Bach and Sorenson[3]. In the Maple programming language, computing $x = a^{1/n}$ can be simplified to

```
irootBachSor:=proc(a,n,flag)
x:=a;   f:= x^n-a;
while 0<f do x:= floor(x-f/(n*x^(n-1)));  f:=x^n-a; od;
if f=0 then assign(flag,true); else assign(flag,false); fi;
x; end;
```

For $a = 10^{1000}$ and $n = 2$, this takes 30 sec. in Maple 6 on a pentium system. Most of this time is due to the poor starting approximation, but even with a starting point with 1 digit correct, the time is 0.6 sec. The main cost being that every iteration works with 1000 digit integers.

Bernstein works with floating point numbers, and progressive precision. Certainly in a Maple context, and perhaps others, progressive precision is more efficiently programmed using floating point numbers rather than integers. Our interest in this algorithm lies in the control of the iterative loop. In progressive precision calculations this is typically programmed as (the program fragment below is not Bernstein's; it is a simplification used for explanation).

```
irootFloat:=proc(a,n,flag) % Digits is global
tol:= Float(1,-Digits); x:=start;
xnew:=0; UserDigits:=Digits; Digits:=1;
while abs(x-xnew)> tol do
Digits:=min(2*Digits,UserDigits); x:= xnew;
xnew:= x- (x^n-a)/(n*x^(n-1)); od;
x; end;
```

This fragment assumes that a start value for x can be computed. The inefficiency with this method is the fact that the last iteration does not improve the accuracy, but simply verifies that the calculation is complete. This is in contrast to p-adic methods in which the number of iterations is known in advance and no extra ones are performed. Using (5) and (6), we have programed Newton and Halley iteration so that no extra iterations are performed.

3.1 Limitations of Newton and Halley iteration

Evaluating (5) for $f = x^n - a$, it can be shown that the improvement in accuracy per step deteriorates with increasing n. In current Maple, the existing `iroot` is faster for $n > 100$, and Halley iteration is faster for $n < 100$.

References

1. D.J. Jeffrey, D.E.G. Hare, and R.M. Corless. Exact rational solutions of a transcendental equation. *C.R. Math. Rep. Acad. Sci. Canada*, 20:71–76, 1998.
2. Daniel J. Bernstein. Detecting perfect powers in essentially linear time. *Mathematics of Computation*, 67:1253–1283, 1998.
3. Eric Bach and Jonathan Sorenson. Sieve algorithms for perfect power testing. *Algorithmica*, 9:313–328, 1993.
4. Richard P. Brent. Fast multiple-precision evaluation of elementary functions. *J. Association Computing Machinery*, 23:242–251, 1976.
5. J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.