High order algorithms to find square roots and inverses

1 Introduction

Inverting a number of size n can be done in time $O(n^2)$ with the classical school division. This time cost can be reduced to O(nlog(n)) when using a fast multiplication, thanks to *Newton's iteration*. The same type of techniques apply to the n-th root computation of a number. For those operations we also describe some very efficient algorithms with high order of convergence (that is quartic, quintic, sextic, ... algorithms). Computing square roots and inverses to a great accuracy is also useful to compute many usual mathematical functions like logarithms and trigonometric functions (see Brent's remarkable paper [2]) or to compute constants like p,log(2),...([2], [6]).

2 High precision inversion

In this section we want to compute the inverse 1/A of the number A to a great accuracy.

2.1 Newton's iteration

Starting from a few digits approximation x_0 of 1/A, we apply *Newton's iteration* to the function f(x) = 1/x-A, which gives the approximating sequence

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = 2x_n - Ax_n^2. \quad (*)$$

It's a property of Newton iteration to converge *quadratically* near the root, that is the number of right digits doubles at each iteration. Thus, a number of about log(n) iterations suffices to have n digits of 1/A.

The iteration (*) have a nice feature : the only needed operations are the product by 2, the difference and the product of large numbers. The Newton convergence being quadratic, the precision needed to compute iteration (*) is not the full final precision but only with the precision wanted at step n .

As an example, we compute the inverse of p, starting from the approximation x_0 = 0.31831 (5 digits). The first three iterations become

$x_1 = 2x_0 - px_0^2$ = 0.3183098861 (operations done with 10 digits of precision)

$x_2 = 2x_1 - px_1^2$ = 0.31830988618379067151 (operations done with 20 digits of precision)

$x_3 = 2x_2 - px_2^2$ = 0.318309886183790671537767526745028724066 (operations done with 40 digit

The first 38 digits of x agree with those of 1/p.

The first 58 digits of x_3 agree with those of 1/p.

The cost of this process is of the order of the multiplication cost : since each iteration involves two multiplications on high precision numbers (that is $x_n \times x_n$ and $A \times x_n^2$), to find 1/A we need to compute 2 multiplications of size n (last step), 2 multiplications of size n/2 (previous step), 2 multiplications of size n/4, etc. Using for example an FFT multiplication, with cost O(nlogn), the final cost is of order

$$2n\log(n)+2\left(\frac{n}{2}\log\left(\frac{n}{2}\right)+2\left(\frac{n}{4}\log\left(\frac{n}{4}\right)+\cdots\right)\right) \le 2n\left(1+\frac{1}{2}+\frac{1}{4}+\cdots\right)\log(n) = 4n\log(n).$$

Note : a better iteration is obtained by writing (*) in the form

$$\begin{aligned} h_n &= 1-Ax_n \\ x_{n+1} &= x_n+x_n h_n. \end{aligned}$$

Since cancellations occurs in $h_n = 1-Ax_n$, the second multiplication $x_n h_n$ can be done by taking into account only the non zero part of h_n. This has also another advantage : the number of vanishing terms in front of h_n gives the approximation precision of x_n.

A division B/A is performed by multiplying B by the inverse of A.

A practical example of inverse of large numbers from FFT multiplication can be found as an easy c sample code at Easy programs for constants computation.

2.2 A cubic iteration

From the general *Householder's iteration* ([3] and [1] p. 216)

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}\left(1+\frac{f(x_n)f''(x_n)}{2f'(x_n)^2}\right),$$

applied on f(x) = 1/x-A, we find

$$\begin{aligned} h_n &= 1-Ax_n \\ x_{n+1} &= x_n+x_n(h_n+h_n^2) \end{aligned}$$

Starting with a good initial point x_0, the convergence is *cubical*, that is, the number of good digits is multiplied by 3 at each step of the process (A similar form of this iteration is given in [1] p. 216 and [4] p. 279). Since h_n tends to zero, the new term h_n^2 can be computed at a lower cost.

As an example, we compute again, the inverse of p, starting from the approximation $x_0 = 0.31831$ (5 digits). The first two iterations are

$$x_1 = 0.3183098861837906715(523...), \text{ (19 good digits)}$$

$$x_2 = 0.318309886183790671537767526745028724068 9...,(58 \text{ good digits})$$

and x_3 has 174 good digits...

2.3 High order iterations

Quartic iteration

It's possible to find a *quartic* iteration, just apply twice Newton's iteration on f(x) = 1/x-A, we find

$$h_n = 1-Ax_n$$

$$x_{n+1} = x_n+x_n(h_n+h_n^2+h_n^3) .$$

If we take a look at our example (inverse of p), starting as usual with $x_0 = 0.31831$, we find that x_1 has 26 good digits, x_2 is 103 digits exact, x_3 has 413 good digits ... The sequence of correct digits is then for this example

$$26,103,413,1650,6601,26405,...$$

Quintic iteration

A *quintic* iteration founded by the application of a general *quintic modified iteration* (here, we omit the details of the demonstration which is deduced from our essay on *Newton's methods*) is given by

$$h_n = 1-Ax_n$$

$$x_{n+1} = x_n+x_n(h_n+h_n^2+h_n^3+h_n^4) = x_n+x_n(1+h_n^2) (h_n+h_n^2) ,$$

the rate of convergence is impressive, again on the previous example, x_1 has 32 correct digits and x_2 has 161 good digits, x_3 has 806 good digits ... The second factorization of this iteration requires only 4 multiplications $(A\times x_n, h_n\times h_n, (1+h_n^2)\times(h_n+h_n^2), x_n\times(1+h_n^2)(h_n+h_n^2))$ at each step!

Higher order

The inverse of a number can be computed with an iteration at any desired order, it's possible to show that the general algorithm of *order* r is given by

$$\begin{aligned} h_n &= 1-Ax_n \\ x_{n+1} &= x_n+x_nP^{r-1}(h_n), \end{aligned}$$

where $P^m(u)$ is a polynomial of degree m, given by

$$\begin{aligned} P^m(u) &= \sum_{k=1}^{m} a_k u^k \\ a_k &= 1. \end{aligned}$$

For example, for m = 3,m = 4, m = 5, m = 6 and m = 7 (respectively quartic, quintic, sextic, septic and octic iterations)

$$\begin{aligned} P^3(u) &= u+u^2+u^3, \\ P^4(u) &= u+u^2+u^3+u^4 = (1+u^2)(u+u^2), \\ P^5(u) &= u+u^2+u^3+u^4+u^5 = u+(1+u)(u^2+u^4), \\ P^6(u) &= u+u^2+u^3+u^4+u^5+u^6 = (1+u^2+u^4)(u+u^2), \\ P^7(u) &= u+u^2+u^3+u^4+u^5+u^6+u^7 = u+(1+u^3)(u^2+u^3+u^4) \\ &... \end{aligned}$$

In practical cases the number h_n tends to zero, therefore the evaluation of the powers h_n^k is more and more efficient when k increases. Observe that some of the given factorizations may reduce the number of required multiplications. A careful implementation of those high order iterations may be very efficient.

3 Square roots

The same technique as for the inverses applies for square roots.

3.1 Newton's iteration

Using the function $f(x) = x^2-A$, Newton iteration gives

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = \frac{x_n}{2} + \frac{A}{2x_n}.$$

(In other words, the iteration is the mean value of x_n and A/x_n).

This iteration has one disadvantage : one should compute the division A/x_n. When one wants to compute $1/\sqrt{A}$, a better iteration is obtained from the function $f(x) = 1/x^2-A$, yielding the iteration

$$x_{n+1} = \frac{3}{2} x_n - \frac{1}{2} A x_n^3. \quad (**)$$

This formula only requires multiplications ! An easier way of computing a square root $\sqrt{A}$ is then to compute $B = 1/\sqrt{A}$ from this latest process and then to perform the product $A \times B = \sqrt{A}$.

Note : as for the inverse, the best way to compute the iteration (**) is to write it in the form

$$h_n = 1-Ax_n^2$$

$$x_{n+1} = x_n + \frac{x_n}{2} h_n,$$

since cancellations occur in $h_n = 1-Ax_n^2$.

3.2 A cubic iteration

Again, from the *Householder's iteration,* applied on the function $f(x) = 1/x^2-A$, we find after some easy manipulations

$$x_{n+1} = \frac{x_n}{8} (15-10Ax_n^2+3A^2x_n^4).$$

This process converges cubically to the inverse of the square root of the number A (This form of the iteration is also given in [1] p. 217). Just like for the quadratic iteration, a more efficient way is to write it

$$h_n = 1-Ax_n^2$$

$$x_{n+1} = x_n + x_n \left(\frac{1}{2} h_n + \frac{3}{8} h_n^2 \right) = x_n + \frac{x_n}{8} (4h_n+3h_n^2) .$$

3.3 High order iterations

Again like for the inverse, it's possible to find a *quartic* iteration, just apply twice Newton's iteration on $f(x) = 1/x^2-A$:

$$x_{n+1} = x_n - \frac{x_n}{16}(1-Ax_n^2)(-20+19Ax_n^2-8A^2x_n^4+A^3x_n^6).$$

This algorithm converges quartically to 1/ÖA.

It is interesting to compare this result to the direct application of the general *quartic modified iteration* to our function (see the essay on Newton's methods)

$$x_{n+1} = x_n - \frac{x_n}{16}(1-Ax_n^2)(-19+16Ax_n^2-5A^2x_n^4).$$

This simpler relation suggests that it may be more efficient to use the quartic iteration than twice the Newton iteration. The best form of this quartic algorithm is

$$h_n = 1-Ax_n^2$$

$$x_{n+1} = x_n + \frac{x_n}{16}(8h_n+6h_n^2+5h_n^3).$$

As an application of the *modified iterations*, we provide the general expression of some high order iterations to compute square roots. The general pattern of a high order iteration of order r is given by

$$h_n = 1-Ax_n^2$$

$$x_{n+1} = x_n+x_nP^{r-1}(h_n)$$

with $P^m(u)$ being a polynomial of degree m with rational coefficients. The first polynomials are given with factorization to save some multiplications, the best choice depends on your implementation.

Quintic iteration

$$P^4(u) = \frac{1}{128}(64u+48u^2+40u^3+35u^4)$$

$$= \frac{1}{128}\left(64u+u^2(48+40u+35u^2) \right) .$$

Sextic iteration

$$P^5(u) = \frac{1}{256}\left(128u+96u^2+80u^3+70u^4+63u^5\right)$$

$$= \frac{1}{256}\left(128u+96u^2+u^3(80+70u+63u^2) \right)$$

Septic iteration

$$P^6(u) = \frac{1}{1024}(512u+384u^2+320u^3+280u^4+252u^5+231u^6)$$

$$= \frac{1}{1024}\left(512u+384u^2+320u^3+u^3(280u+252u^2+231u^3) \right)$$

Octic iteration

$$P^7(u) = \frac{1}{2048}(1024u+768u^2+640u^3+560u^4+504u^5+462u^6+429u^7)$$

$$= \frac{1}{2048}(1024u+768u^2+640u^3+560u^4+u^4(504u+462u^2+429u^3))$$

For example to evaluate $P^7(u)$, you may compute in this order

$$u\times u, u\times u^2, u\times u^3, u^4(504u+462u^2+429u^3)$$

that is, four large multiplications and don't forget that, in our cases, u is small so that those multiplications are even faster to perform.

Expression of the polynomials $P^m(u)$

The coefficients of the polynomials $P^m(u)$ are in fact given by the series expansion of

$$\frac{1}{\sqrt{1-u}} - 1 = \sum_{k=1}^{\infty} a_k u^k$$

$$a_k = \frac{2(2k-1)!}{k!(k-1)!4^k}$$

this allow to define easily any order algorithm to compute square roots.

4 m-th roots

The previous techniques may be used in order to compute the m-th root (m being an integer) of the number A. Therefore, to find with a great accuracy the number $A^{-1/m}$, we use Newton's method on the function $f(x) = 1/x^m - A$ and this will produce the process

$$h_n = 1 - A x_n^m$$

$$x_{n+1} = x_n + x_n \frac{h_n}{m}$$

which converges quadratically to $A^{-1/m}$. To find $A^{1/m}$ just compute at the end of the N previous iterations

$$y_N = A x_N^{m-1}.$$

Householder's iteration also becomes

$$h_n = 1 - A x_n^m$$

$$x_{n+1} = x_n + x_n \left(\frac{1}{m} h_n + \frac{1+m}{2m^2} h_n^2 \right)$$

which converges cubically to $A^{-1/m}$.

4.1 High order iteration

The general pattern for any order iteration is given by

$$h_n = 1-Ax_n^m$$

$$x_{n+1} = x_n+x_nP(h_n)$$

and to find P(u) you need to compute the following series expansion

$$(1-u)^{-1/m}-1 = \frac{1}{m}u+\frac{1+m}{2m^2}u^2+\frac{(1+m)(1+2m)}{3!m^3}u^3+...$$

The truncated series at order k will provide an algorithm of order k+1. Observe that for the value m = 1, we retrieve the polynomials used to compute the inverse of a number and for m = 2, it gives back the algorithms to find square roots.

4.1.1 Example

For m = 4 the following algorithm converges quartically to $A^{-1/4}$

$$h_n = 1-Ax_n^4$$

$$x_{n+1} = x_n+x_n\left(\frac{1}{4}h_n+\frac{5}{32}h_n^2+\frac{15}{128}h_n^3\right)$$

and $A^{1/4}$ is given by

$$Ax_n^3.$$

References

[1] J.M. Borwein and P.B. Borwein, *''Pi and the AGM - A study in Analytic Number Theory and Computational Complexity''*, A Wiley-Interscience Publication, New York, (1987)

[2] R.P. Brent, *Fast multiple-Precision evaluation of elementary functions*, J. Assoc. Comput. Mach., (1976), vol. 23, p. 242-251

[3] A. S. Householder, *The Numerical Treatment of a Single Nonlinear Equation*, McGraw-Hill, New York, (1970)

[4] D.E. Knuth, *The Art of Computer Programming, Vol. II, Seminumerical Algorithms,* Addison Wesley, (1998).

[5] I. Newton, *Methodus fluxionum et serierum infinitarum*, (1664-1671)

[6] E. Salamin, *Computation of* p *Using Arithmetic-Geometric Mean*, Mathematics of Computation, (1976), vol. 30, p. 565-570 This is why it's important to find such algorithms.

File translated from T_EX by T_TH, version 2.32.

On 5 Feb 2001, 19:06.