Square Root Without Division

The objective is to compute $Y := \sqrt{X}$ for a given positive X without using division operations. All schemes for doing so exploit approximations r to $R := 1/Y = 1/\sqrt{X}$. Such approximations can be improved arbitrarily, limited only by roundoff, via the *Reciproot Iteration* :

Given a not too bad approximation $r \approx 1/\sqrt{X}$,
a better approximation is $\bar{r} := r + (1 - Xr^2)r/2$.

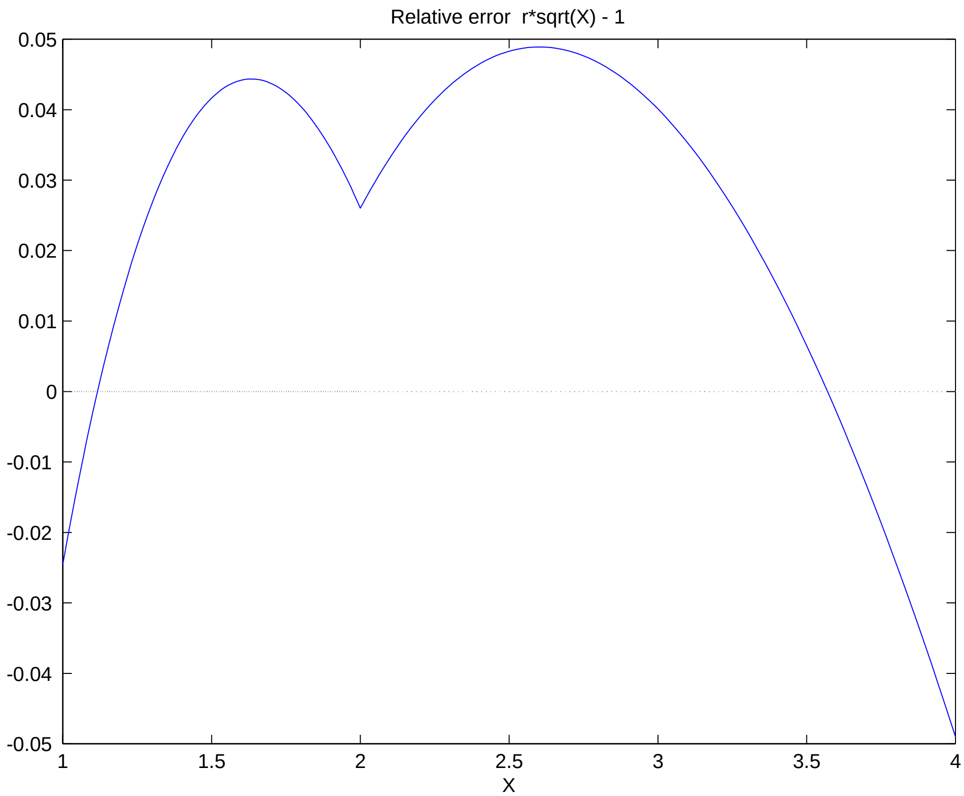
“ Not too bad ” means $0 < r\sqrt{X} < \sqrt{3}$, and then $\bar{r}$ has almost twice as many correct sig. bits as r has; $\bar{r}\sqrt{X} - 1 = -(r\sqrt{X} - 1)^2(r\sqrt{X} + 2)/2$. Each quadratically convergent Reciproot iteration costs two add/subtractions and three or four multiplications, depending upon how multiplication by $1/2$ is implemented. Repeated iteration until R has been approximated adequately yields an adequate approximation to $\sqrt{X} = RX$ too at the cost of another multiplication. Combining this multiplication with the last iteration for $\bar{r}$ to improve $y := rX$ to $\bar{y} := \bar{r}X = y + (X - y^2)r/2$ saves a multiplication and, if r is accurate enough, provides a final $\bar{y} \approx \sqrt{X}$ almost correctly rounded.

Quadratically convergent Reciproot iteration costs more per iteration than a linearly convergent iteration that uses one fixed approximate $r \approx 1/\sqrt{X}$ to improve each of a sequence of unrelated approximations $y \approx \sqrt{X}$ to $\bar{y} := y + (X - y^2)r/2$ at the cost of two add/subtractions and two or three multiplications per iteration. Provided r is close enough to $1/\sqrt{X}$ and y is close enough to $\sqrt{X}$, the new relative error $\bar{y}/\sqrt{X} - 1 = -(y/\sqrt{X} - 1)\big((y/\sqrt{X} - 1) + (y/\sqrt{X} + 1)(r\sqrt{X} - 1)\big)/2$ will be smaller than the old; each repeated iteration will gain about as many correct sig. bits for $\bar{y}$ as r has. This linearly convergent iteration makes sense when the ultimate accuracy desired is not much better than has already been achieved in y .

A first approximation $r \approx 1/\sqrt{X}$ is constructed via a small table-look-up. Except for special cases like $X = \infty$, $X = 0$ and subnormal X , IEEE 754 formatted $X = 2^k(1 + \cdot f)$ is stored in a floating-point word whose fixed-point interpretation is $Z = (k+B) + \cdot f$, where $0 \le \cdot f < 1$ and B is the exponent bias, an integer like k . A fixed-point constant $C+\cdot g$ slightly less than $3B/2$ can be so chosen, with integer C and fraction $\cdot g$ lined up around the “ binary point ” just like Z , that a shift and subtract produce the fixed-point word $S := C+\cdot g - Z/2$ whose floating-point interpretation is the desired first approximation r good to at least three sig. bits. For instance, when $B = 127$, set $C+\cdot g := 190.451$ to keep the relative error in r within ± 0.05 .

For better accuracy, some bits of $\cdot g$ should be taken from a table indexed by a few bits of Z including the last bit of $k+B$ and the leading few bits of $\cdot f$. Every additional bit of index more than doubles the memory bits needed by the table and contributes about one additional sig. bit to r , whose accuracy will be multiplied by subsequent Reciproot and other iterations. The optimal values for $\cdot g$ depend somewhat upon the way in which r will figure in subsequent iterations. Tricky details, including how to get $\sqrt{X}$ correctly rounded at the end, must be left to another occasion.

```
function e = rcprtplt(g, N)   % ...  written for  MATLAB
%  To estimate  1/sqrt((1+f)*2^j)  for  j = 0  or  1  and  0 <= f < 1 ,  try
%  the approximation  r = (3/2 + g - j/2 - f/2)/2  for some  0 < g <= 1/2 .
%  Rcprtplt(g)  plots r ‘s relative error as a function of  g  at  2^N  points.
%  ( By default,  N = 8 .)  A good value for  g  is  0.451 .   ((C) W. Kahan)
if  ( g <= 0 | g > 0.5 ) ,  g,  error('Keep  0 < g <= 0.5 .'),  end
if  nargin < 2 ,  N = 8 ;  end
N = round(N) ;        % ...   Make sure  N  is an integer.
n = 2^(N-1) ;  f = [0: n]'/n ;  x = 1+f ;  x = [x(1:n); 2*x] ;
r1 = 0.5*( 1 + g - 0.5*f ) ;
r0 = r1(1:n) + 0.25 ;
r = [r0; r1] ;        % ...   r approximates  1/sqrt(x)
e = r.*sqrt(x) - 1 ;  % ...   1 <= x = (1 + f)*2^j < 4
plot(x, e, x, 0) ;
title('Relative error  r*sqrt(X) - 1') ;
xlabel('X') ;
```



Reciproot Iterations of Higher Order:

For an iteration of order $k \ge 2$ let $q_k(z)$ be the polynomial in z obtained from the first k terms of the Taylor series

$$(1-z)^{-1/2} = 1 + z/2 + 3z^2/8 + 5z^3/16 + 35z^4/128 + 63z^5/256 + 231z^6/1024 + 429z^7/2048 + \dots$$

so that $q_k(z) = (1-z)^{-1/2} \pm O(|z|^k)$. Then the iteration that replaces $r \approx 1/\sqrt{X}$ by

$$\bar{r} := r \cdot q_k(1 - Xr^2) = 1/\sqrt{X} \pm O(|1 - Xr^2|^k)/\sqrt{X}$$

is an iteration of order k . Implemented in floating-point, each such iteration costs $k+2$ multiplications and k add/subtractions. This implies that the iteration's efficiency is *ultimately* best when the order k minimizes

$$\big(\text{ time for } (k+2) \text{ multiplications and } k \text{ add/subtractions } \big)/\log(k) ,$$

which occurs when k is 2, 3 or 4 , depending upon the relative costs of multiplication and addition/subtraction. For example, one iteration with $k = 9$ replaces relative error $1 - r\sqrt{X}$ by $1 - \bar{r}\sqrt{X} \approx \pm O(|1 - r\sqrt{X}|^9)$ at the cost of eleven multiplications and nine add/subtracts; but two iterations with $k = 3$ make a roughly similar reduction in the relative error (if it's small enough) at the lower cost of ten multiplications and six add/subtractions. However, M. Keynes said "*ultimately* we are all dead"; so the optimal order k may be determined by other considerations when the relative error in r is not very tiny. Anyway, order $k > 4$ seems implausible.

Further Reading

Articles about computing, rounding and testing square roots will appear in the Proceedings of the 14th IEEE Symposium on Computer Arithmetic to be held in Adelaide, Australia, 14-16 April 1999. Until then many of these articles can be found posted at

http://www.ecs.umass.edu/ece/arith14/program.html