

Singer on the Root

Jack Powles

The specialised fast square root recipe proposed by Konrad Singer (CCP5 Inf. Quart. March 1983, p.47) has, it appears, been very popular and rightly so. It is based on the least squares polynomial fit for, say four terms to square root of x from x=0.1 to 1.0,

$$\text{root } x \approx a_0 + x(a_1 + x(a_2 + x a_3)).$$

However the values of the a_i given by Singer are wrong! The correct values are, I hope (Singer's in brackets),

$$\begin{aligned} a_0 &= 0.188030699 & (0.1882532) \\ a_1 &= 1.48359853 & (1.428264) \\ a_2 &= -1.0979059 & (-1.097209) \\ a_3 &= 0.430357353 & (0.430526) \end{aligned}$$

The fit to root x, x=0.1,0.05,1.0, now has a rms devn. of 0.0028 (0.0333).

If users have not noticed this DO NOT PANIC.

Singer's iterations soon give the right answer to any desired accuracy, even on the CDC which gave the wrong values for the a_i ! However with the correct parameters at least one fewer iteration is needed, depending on your machine. Since the whole object is to make it fast this is surely worth having, so change your BLOCK DATA.

This routine is so fast it is difficult to time. It is no good, for instance using a DO loop i=1,10000, say, because the looping overheads swamp it. In fact I do not know how to do it. Maybe our kind Editor will add the answer to the end of this Note - if they still have a computer in Daresbury?

Footnote

Yes we do indeed still have a computer at Daresbury, though not alas, the Cray-1. Presently we 'make do' with the NAS AS 7000 which has replaced our departed IBM 370/165.

The problem posed by Professor Powles is not a trivial one. It is surprisingly difficult to extract accurate values for the execution times for a given routine from amongst the many operations that occur in even the simplest of jobs. Such an evaluation must inevitably be system dependent and I confess I know of no general way of obtaining an accurate one. However, in the usual hand-waving way that computational (as opposed to computer) scientists work, a rough estimate is possible.

To prevent the swamping of the square-root timing by the DO-loop overheads I have evaluated 256 double precision square-roots in the same 1000 step loop. I have also averaged the results over 10 runs, to obtain a figure less susceptible to system vagaries. The time for the execution of the corresponding 'empty loop' was similarly obtained and subtracted. The system SQR routine averaged 4.705 seconds for 256,000 square roots. The Singer/Powles algorithm (which I call SPSQR) averaged 5.173 seconds for the same number (assuming two cycles of the Newton-Raphson procedure).

This result is not too encouraging until one realises that it is not necessary to use the SPSQR routine as an external function (as is the system SQR). Inserting the relevant code in-line, at the point where the square root is required, removes the overheads associated with an external function call. Using SPSQR in this manner gave a result of 2.004 seconds for our 256,000 square roots. Such a substantial improvement over the system SQR routine is not to be ignored!

On the subject of accuracy, I have compared the Singer and Powles algorithms (each assuming two cycles of the Newton-Raphson iteration) with the system SQR routine. Assuming the SQR routine to be 100% accurate, the Singer algorithm gave an average error of $-4E-6$ with a RMS deviation about this error of $4E-6$. The averages being obtained from a determination of 10,000 square roots in the range 0.1 to 1.0. The Powles algorithm was noticeably better, with an average error of $-2E-9$ and a RMS deviation about this error of $2E-8$.

Lastly, assuming that the Powles formulation of the Singer algorithm with two Newton-Raphson iterations is acceptably accurate, the algorithm can be made yet more efficient if the two Newton-Raphson steps are with as an explicit formula thus:-

$$Y = (.430357353 * X - 1.0979059) * X + 1.48359853 * X + .188030699$$

$$Z = .25 * (Y + X/Y) + X / (Y + X/Y)$$

where $Z = \sqrt{X}$



This algorithm then requires 1.543 seconds to evaluate our 256,000 square roots.

W. Smith