Root Approximation ver 2.1

History of Square Root Calculation

No one knows who invented the square root, but it is thought that the knowledge of square roots originally came from dividing areas of land into equal parts so that the length of the side of a square became the square root of its area, Pythagoras' theorem (5th century BCE), when applied to a right-angled triangle whose sides are 1 unit in length, yields a hypothenuse whose length is equal to square root of 2. Thus, square root of 2 is a number arising as a measure of length of a line segment. The discovery that such a number is not a ratio of whole numbers created a crisis of enormous magnitude for the Pythagoreans. On one hand, it invalidated many of their geometric proofs, which relied heavily on the assumption that lengths of line segments were rational numbers; and on the other hand, it shattered their deeply held belief in the supremacy of whole numbers as the underlying principle of the universe. In addition, Hippasus, one of Pythagoras' students, breached their most sacred rules of conduct, he revealed his discovery of the irrational number square root of 2 thereby breaking his oaths of both secrecy and individuality. For his sins, legend has it, he was thrown overboard during a sea voyage. Euclid is known as the Father of Geometry. He lived several years after Pythagoras, and he continued the work of Pythagoras. Euclid focused mainly on the right angle 3:4:5 ratio puzzle. Pythagoras and Euclid play a significant symbolic role in Freemasonry.

Before Pythagorus, the Babylonians and Greeks have been credited with the discovery of Heron's square root method, the precursor of Newton's iterative method, although Indian mathematicians are thought to have used a similar system around 800BC. The Egyptians calculated square roots using an inverse proportion method as far back as 1650BC. Chinese mathematical writings from around 200BC show that square roots were being approximated using an excess and deficiency method. In 1450AD Regiomontanus invented a symbol for a square root, written as an elaborate R. The square root symbol √ was first used in print in 1525.

Computers have popularized recursive or iterative square root algorithms, such as Newton's method, which start with an approximation, or guess, of the square root and find the higher order digits first. Such iterative methods can be carried out on a computer, but they are usually difficult to implement for very large numbers and computational difficulty can arise with the division operation.

The principal square root of most numbers is an irrational number with an infinite decimal expansion. As a result, the decimal expansion of any such square root can only be computed to some finite-precision approximation. However, even if we are taking the square root of a perfect square integer, so that the result does have an exact finite representation, the procedure used to compute it may only return a series of increasingly accurate approximations.

The most common analytical methods are iterative and consist of two steps: finding a suitable starting value, followed by iterative refinement until some termination criterion is met. The starting value can be any number, but fewer iterations will be required the closer it is to the result. The most familiar such method, most suited for programmatic calculation, is Newton's method, which is based on a property of the derivative in the calculus. A few methods like paper-and-pencil synthetic division and series expansion, do not require a starting value. In some applications, an integer square root is required, which is the square root rounded or truncated to the nearest integer (such as **Heron's method**, after the first-century Greek mathematician Hero of Alexandria who described the method in his AD 60 work *Metrica*). Today, nearly all computing devices have a fast and accurate square root function.

Heron's method from first century Egypt was the first ascertainable algorithm for computing square root. Heron's method is a first order approximation and can be considered to be

$$\sqrt{x} \cong a + \frac{b}{2a}$$

Where a^2 is the closest perfect square to x
and $b = x - a^2$. Thus $\sqrt{x} = \sqrt{a^2 + b}$ and the root is fairly accurate for b much smaller than a.

If you are a teacher instructing students in the fine art of object oriented programming, as a programming exercise, here is a question you could ask: "is it possible to perform common roots in PHP without using the built-in function or using the traditional iterative methods such as "Newton's Method". The answer is yes.

A non-iterative extension or refinement of Heron's method follows below.

Non-Iterative Square Root Approximation

$$\sqrt{x} = a + \frac{b}{2a} * (1 - 2ndO) + 3rdO$$

Where a is the closest integer square root to x and $b = x - a^2$ thus $\sqrt{x} = \sqrt{a^2 + b}$

The first order approximation is

$$\sqrt{x} = a + \frac{b}{2a}$$

The 2^{nd} order approximation adds term $2ndO = \frac{b}{d}$

with $\quad d = 4a^2 + 2b - \left(\frac{b}{2a+1}\right)$

Then letting the 2nd order approximation equal = “u” for $\sqrt{x}$ then the second order approximation is

$$u = a + \frac{b}{2a} * \left(1 - \frac{b}{4a^2 + 2b - \left(\frac{b}{2a+1}\right)}\right)$$

The 3rd order approximation term is

$$v = \frac{x - u^2}{2 * u}$$

adding the 3rd order term results in this eqn

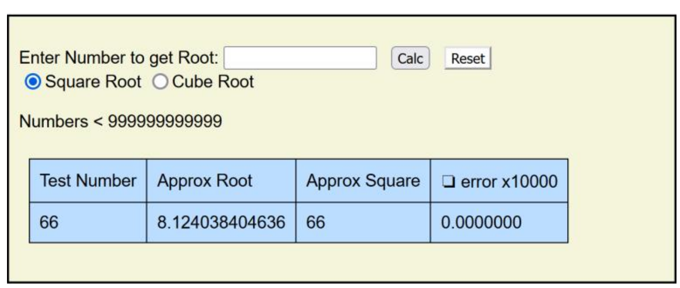
$$\sqrt{x} = u + v \quad \text{or} \quad \sqrt{x} = u + \frac{x - u^2}{2u}$$

$$\text{or} \quad \boxed{\sqrt{x} \cong \frac{x + u^2}{2u}}$$

with a typical error of less than 0.000000001 and

a maximum error of 0.0000002165 for $\sqrt{12}$.

Using simple PHP code, the algorithm was tested for various small and large numbers as listed below. A test sample of a hundred numbers was executed in less than a second. The PHP code is listed in the appendix. An online demonstration is available from the archive : https://tinyurl.com/58wzdskc



Test Number	Approx Root	Approx Square	❏ error x10000
66	8.124038404636	66	0.0000000

Square Root Algorithm Test Results			
Test Number	Approx Root	Approximate Square	Square error x1000:
0.0144	0.12	0.0144	0.0000000
0.0145	0.12041594578792	0.0145	0.0000000
0.015	0.12247448713916	0.015	0.0000000
0.0155	0.12449899597989	0.0155	0.0000000
0.016	0.12649110640674	0.016	0.0000000
0.0165	0.12845232578665	0.0165	0.0000000
0.0169	0.13	0.0169	0.0000000
9	3	9	0.0000000
10	3.1622776601972	10.0000000002	0.0000018
11	3.3166247912733	11.0000000061	0.0000609
12	3.4641016463851	12.0000002165	0.0021649
13	3.6055512791487	13.0000000266	0.0002657
14	3.7416573869881	14.0000000016	0.0000160
15	3.8729833462096	15	0.0000002
16	4	16	0.0000000
17	4.1231056256186	17	0.0000001
18	4.2426406871564	18.0000000003	0.0000031
19	4.358898943797	19.0000000022	0.0000223
20	4.4721359607024	20.000000051	0.0005101
21	4.5825756960792	21.0000000103	0.0001030
22	4.6904157599731	22.0000000014	0.0000140
23	4.7958315233225	23.0000000001	0.0000009
24	4.8989794855665	24	0.0000000
25	5	25	0.0000000
26	5.0990195135928	26	0.0000000
27	5.1961524227093	27	0.0000003
28	5.2915026221507	28.0000000002	0.0000023
29	5.3851648072162	29.0000000009	0.0000088
30	5.4772255765088	30.000000016	0.0001596
31	5.5677643632236	31.0000000044	0.0000438
32	5.6568542495753	32.0000000009	0.0000094
33	5.7445626465498	33.0000000001	0.0000014
34	5.8309518948461	34	0.0000001
35	5.9160797830996	35	0.0000000
36	6	36	0.0000000
37	6.0827625302982	37	0.0000000
38	6.1644140029693	38	0.0000000
39	6.244997998401	39	0.0000003
40	6.3245553203476	40.0000000001	0.0000014
41	6.4031242374628	41.0000000004	0.0000038
42	6.4807406988738	42.000000006	0.0000604
43	6.5574385244578	43.000000002	0.0000204
44	6.6332495807547	44.0000000006	0.0000058
45	6.708203932509	45.0000000001	0.0000013
46	6.7823299831267	46	0.0000002
47	6.8556546004011	47	0.0000000
48	6.9282032302755	48	0.0000000
49	7	49	0.0000000
50	7.0710678118655	50	0.0000000
51	7.1414284285429	51	0.0000000
52	7.2111025509284	52	0.0000001
53	7.2801098892824	53	0.0000003
54	7.348469228355	54.0000000001	0.0000008
55	7.416198487108	55.0000000002	0.0000018
56	7.4833147737229	56.0000000026	0.0000262
57	7.549834435339	57.000000001	0.0000103
58	7.6157731058874	58.0000000004	0.0000036
59	7.6811457478754	59.0000000001	0.0000010
60	7.7459666924164	60	0.0000002
61	7.8102496759069	61	0.0000000

62	7.8740078740118	62	0.0000000
63	7.9372539331938	63	0.0000000
64	8	64	0.0000000
65	8.0622577482985	65	0.0000000
66	8.124038404636	66	0.0000000
67	8.1853527718725	67	0.0000000
68	8.2462112512357	68	0.0000001
69	8.3066238629193	69	0.0000002
70	8.3666002653436	70	0.0000005
71	8.4261497731819	71.0000000001	0.0000009
72	8.4852813743127	72.0000000013	0.0000126
73	8.54400374535	73.0000000006	0.0000055
74	8.6023252670555	74.0000000002	0.0000022
75	8.6602540378489	75.0000000001	0.0000008
76	8.7177978870827	76	0.0000002
77	8.7749643873924	77	0.0000001
78	8.8317608663279	78	0.0000000
79	8.8881944173156	79	0.0000000
80	8.9442719099992	80	0.0000000
81	9	81	0.0000000
82	9.0553851381374	82	0.0000000
83	9.1104335791443	83	0.0000000
84	9.1651513899117	84	0.0000000
85	9.219544457293	85	0.0000000
86	9.273618495496	86	0.0000001
87	9.3273790530895	87	0.0000001
88	9.3808315196484	88	0.0000003
89	9.4339811320593	89.0000000001	0.0000005
90	9.4868329805397	90.0000000007	0.0000066
91	9.539392014186	91.0000000003	0.0000032
92	9.5916630466328	92.0000000001	0.0000014
93	9.6436507609959	93.0000000001	0.0000006
94	9.6953597148337	94	0.0000002
95	9.7467943448093	95	0.0000001
96	9.7979589711328	96	0.0000000
97	9.8488578017961	97	0.0000000
98	9.8994949366117	98	0.0000000
99	9.9498743710662	99	0.0000000
100	10	100	0.0000000
101	10.049875621121	101	0.0000000
102	10.099504938362	102	0.0000000
103	10.148891565092	103	0.0000000
104	10.198039027186	104	0.0000000
105	10.24695076596	105	0.0000000
106	10.295630140987	106	0.0000000
107	10.344080432789	107	0.0000001
108	10.392304845414	108	0.0000002
109	10.440306508912	109	0.0000003
110	10.488088481719	110.0000000004	0.0000036
111	10.535653752862	111.0000000002	0.0000019
112	10.583005244263	112.0000000001	0.0000009
113	10.630145812737	113	0.0000004
114	10.677078252032	114	0.0000002
115	10.723805294764	115	0.0000001
116	10.770329614269	116	0.0000000
117	10.816653826392	117	0.0000000
118 No significant errors > 0.0000000001 beyond here			

The Curious Route 66

"**Get Your Kicks on Route 66**"* is a popular rhythm and blues song, composed in 1946. The lyrics relate to a westward road trip on U.S. Route 66, a highway which traversed the western two-thirds of the United States from Chicago, Illinois, to Los Angeles, California. The song became a standard, with several renditions appearing on the record charts. It later spawned the 1960s TV series "Route 66" that featured Martin Milner as Tod and George Maharis as Buz and a C1 Corvette. The two young adventurers drove the road in their Corvette for 116 episodes which aired over four seasons.

The exact root of 66 = 8.124038404635...

The root of 66 can be shown to be exactly equal to:

$$\sqrt{66} = 8 + \cfrac{1}{8 + \cfrac{1}{8 + \sqrt{66}}} \qquad (4)$$

Since eqn (4) is an equality there is no error, however it can not be solved due to the root on the right hand side of the equation, but a first order approximation from eqn (2) can be used:

$$\sqrt{66} \cong a + \frac{b}{2a}$$

Where a = 8 and b = 2, the approximation = 8 + 2/16 = 8 + 1/8

- Substituting this into equation (4) for the right hand side root results in the approximate value of the root of 66 as 8.124038462 vs. the exact value of 8.124038404635. An error of just 0.000000057. And the approximate square = 66.000000093.

Using the third order approximation from the first section

$$\sqrt{x} \cong \frac{x+u^2}{2u}$$

yields an accurate approximation of $\sqrt{66}$ as 8.124038404636. The approximate square is 66.0000000000064.

You can now get your kicks with root 66 !.

* Nat King Cole: https://www.youtube.com/watch?v=ikwPxniT1Rw

Non-Iterative Cube Root Approximation

The cube root approximation follows a similar approach to that of the square root approximation.

1. Find the closest integer cube

2. Find the first order approximation where a^3 is the integer cube for

$$\sqrt[3]{x} = \sqrt{a^3 + b}$$

and

$$\sqrt[3]{x} \sim a + \frac{b}{3 * a * a}$$

3. Then tailoring it using the following logic:

a = Integer Root;

b = offset;

Cube = a * a * a + b;

q = b / (3 * a * a);

F = a + q; // first order approximation

G = a + (F * q / (F + q));

H = (Cube + G^3) / (2 * G*G));

J = (G + H) / 2;

K = (Cube +J^3) / (2 * J*J);

Approximate Cube Root = (2 * K + (Cube / K * K)) / 3;

Non-Iterative 5th Root Approximation

The fifth root approximation follows a similar approach to that of the cube root approximation.

1. Find the closest integer fifth power

2. Find the first order approximation where a^5 is the integer 5^{th} for

$$X^{1/5} = (a^5 + b)^{1/5}$$

and

$$X^{1/5} \sim a + \frac{b}{5*a*a*a*a}$$

2. The difficulty here is that “a” must be significantly bigger than “b” for this to work. And for small values of “x”, a work around is required by multiplying “x” by a big constant, finding the root, then dividing by the constant to acquire the answer. Five to the 5th power (3125) is a convenient constant. In addition, if “b” is larger than about half the distance to the next large integer 5^{Th} power then stepping backwards from there vs forward from the lower integer 5^{th} power improves accuracy.

3. After multiplying by the constant, then using the following logic (PHP powers are denoted by pow(num,pow) here we are using carat notation: num^pow):

a = Integer Root;

b = offset;

fifthPow = 5^{th} power = a * a * a*a*a + b;

q = b / (5 * a * a*a*a);

(FO = a + q; // first order approximation);

However "N" is a better first order approximation for 5th root:

N = a + (1 / ((1 / q) + (1 / ((a + q) / 2))));

R = a + (q * N) / (q + N);

S = (fifthPow + R^5) / (2 *R^4);

T = (R + S) / 2;

U = (fifthPow + T^5) / (2 * T^4);

V = ((4 * U) + (fifthPow / U^4)) / powerDivisor;

W = V / 5;

Y = a + (1 / ((1 / q) + (1 / ((U) / 2))));

Z = Y /powerDivisor;

AprxRoot = (W + Z) / 2;

TempVar= (AprxRoot*powerDivisor);

Approximate 5th power = (TempVar)^5;

Errorfix = (fifthPow - Approximate 5th power)/(5* TempVar^4)

Approximate 5throot = AprxRoot + Errorfix/powerDivisor ;

See the PHP files for specific implementation.

Alternate Source for PHP CODE:
See URL with attached php files:
classRoot66.php
Root66Implementation.php
URL: https://tinyurl.com/58wzdskc