Title: A New Perspective on Numerical Trigonometric Approximations
Author: A. Eftimiades
Comments: 15 Pages; 10 jpg Figures
Subj-class: History and Overview
MSC-class: 65D15 (Primary), 00A08 (Secondary) 97A20
Abstract:

As a high school student, I attempted to find a numerical approximation to the six trigonometric functions and their inverses. The result I derived is a generalized variation of the Archimedes Pi Approximation. The numerical algorithms themselves have several intriguing connections to other fields, such as logistics, and rational trigonometry.

$$2^n f^n(x) \leq \pm \sin^{-1}(x) \leq 2^n \frac{f^n(x)}{\sqrt{1 - f^n(x)^2}}$$

$$f(x) = \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - x^2}\right)}$$

$$s^n\left(\frac{x}{\sqrt{4^n + x^2}}\right) \leq \pm \sin(x) \leq s^n\left(\frac{x}{2^n}\right)$$

$$s(x) = 2x\sqrt{1 - x^2}$$

To better understand the preceding text, it is important to understand Archimedes' Pi approximation (which is actually a special case of this algorithm.) This is the simplest approach to approximating Pi and is done with polygons. A circle can be considered a polygon with an infinite number of sides (shown below.)

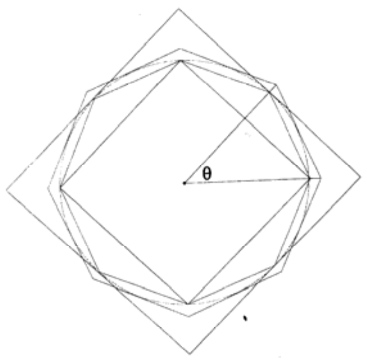


Fig 1a

The Lower Limit

The following method, although similar to the approach Archimedes took in approximating Pi, is more general because it can define any angle. (It also offers interesting geometric interpretations.)

If we consider circumscribing a right triangle ABC (see below)

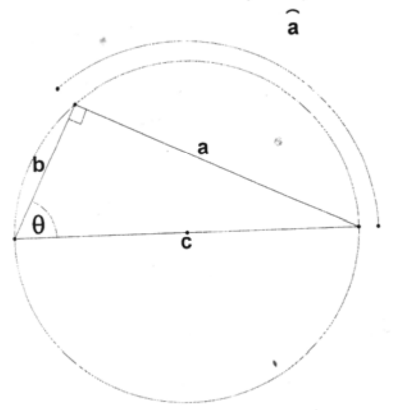


Fig 1b

c is the diameter and **a** and **b** are chords. θ is angle A and, when measured in radians, is the length of the arc adjacent to a, which we will call $\hat{a}$ **.** In all respects, $\hat{a}$ is the same as θ, when measured in radians, but specifically refers to the arc projected by θ.

This next drawing is simply derived from a few corollaries:

-The radius (or **c**/2 because **c** is the diameter) can be pivoted about the center of the circle.

-The radius of a circle **c**/2, when at the correct angle, can always be a perpendicular bisector to any chord (such as **a.**)

-The Side Splitting Theorem (which shows that x/**b**=(**c**/2)/ **c**), or in other words, x=**b**/2)

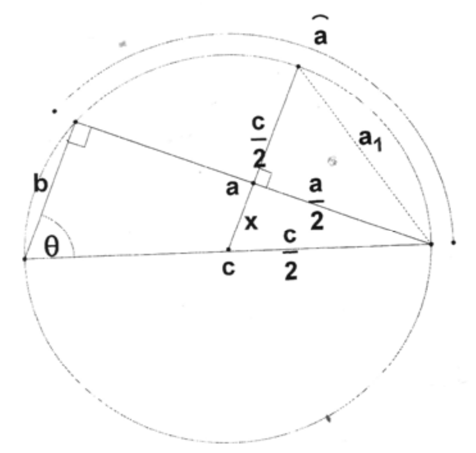


Fig 2

unknown segments are shown with dotted lines

Now we tie everything together. Since the radius of a circle is always a perpendicular bisector to a chord, we can use the Pythagorean Theorem to create a new chord. We will call this new chord a_1.

$$a_1 = \sqrt{\left(\tfrac{c}{2} - \tfrac{b}{2}\right)^2 + \left(\tfrac{a}{2}\right)^2}$$

Now let's simplify.

$$\sqrt{\tfrac{1}{4}\left(c^2 + b^2 + a^2 - 2cb\right)}$$

Using the Pythagorean Theorem, we can substitute **b** for $\sqrt{c^2 - b^2}$ to obtain an expression in terms of **c** and **a**.

$$\sqrt{\tfrac{1}{4}\left(2c^2 + 2c\sqrt{c^2 - a^2}\right)}$$

$$a_1 = \sqrt{\tfrac{1}{2}\left(c^2 - c\sqrt{c^2 - a^2}\right)}$$

Now, if we just move our radius **c**/2 over until it is the perpendicular bisector of our new chord a_1, we can do the exact same process with **c**/2 and a_1/2 as we did with **c**/2 and **a**/2. We will call this new chord a_2.

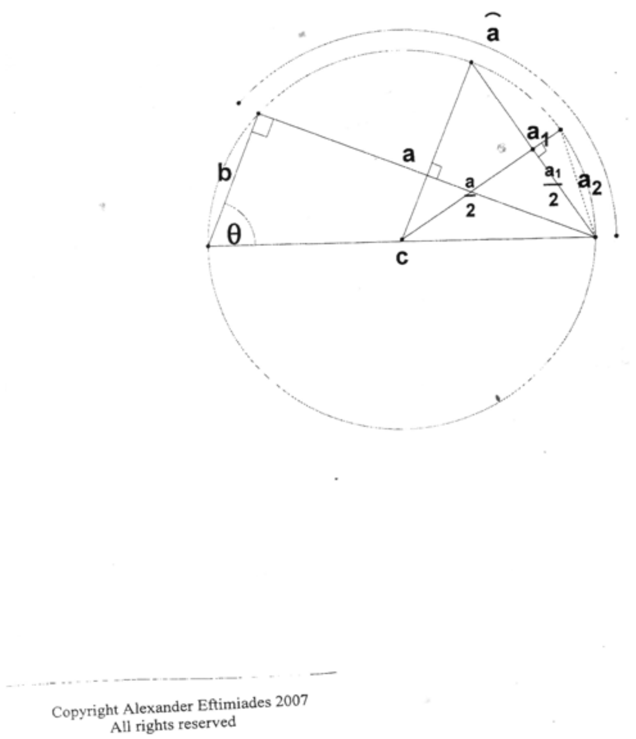


Fig 3

$$a_2 = \sqrt{\tfrac{1}{2}\left(c^2 - c\sqrt{c^2 - a_1^2}\right)}$$

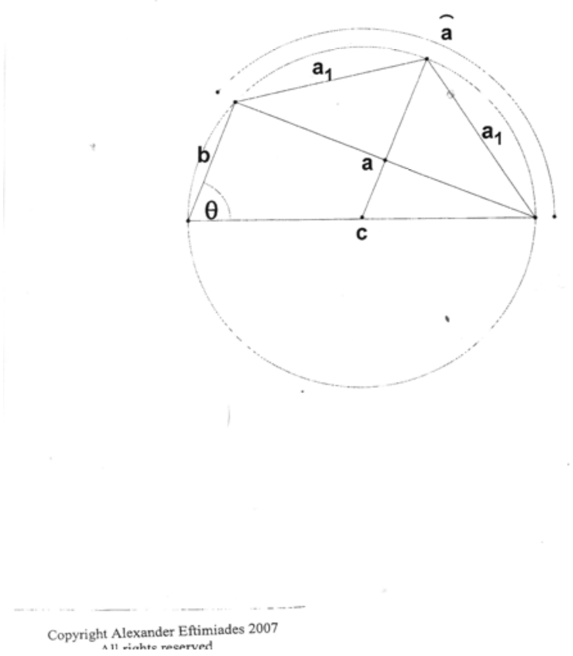
See the pattern? Just to write it out . . .

$$a_{n+1} = \sqrt{\tfrac{1}{2}\left(c^2 - c\sqrt{c^2 - a_n{}^2}\right)}$$

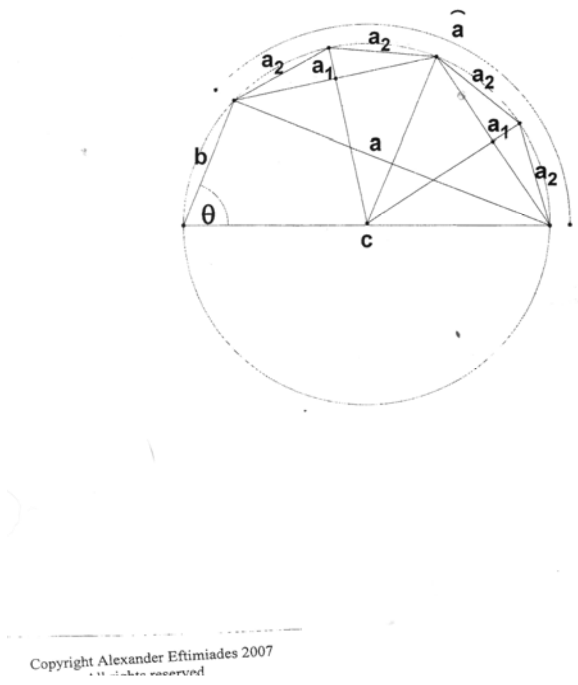
. . . such that a_0 is the original leg **a** of the original triangle ABC.

These will be used in the same manner to approximate an angle as the lengths of the sides of the polygons seen in the Archimedes Pi approximation were used to define Pi. If a_1 is doubled, it is comparable to $\widehat{a}$ . .

Fig 4



. . . and when a_2 is quadrupled, it is very close to $\widehat{a}$.



If we keep repeating this iteration, the value will approach $\hat{a}$, thus replicating inverse sin (since we found θ in terms of **a** and **c**.) Before moving on, the algorithm must be adjusted to be in terms of **a**/**c**. To do this, we must remember both **a** and **c** are measurements; be it in meters, centimeters, feet, or the like, lengths **a** and **c** are both relative to some unit length. But what if the unit length were **c** itself? Then **a** would be measured in terms of **c** (becoming **a**/**c**) and **c** would be 1 (because it is the unit length.)

So let's see what our final formula for $\left(\frac{a}{c}\right)_n$ would be.

$$\left(\tfrac{a}{c}\right)_{n+1} = \sqrt{\tfrac{1}{2}\left(1^2 - 1\sqrt{1 - \left(\tfrac{a}{c}\right)_n^{\ 2}}\right)}$$

or

$$\left(\tfrac{a}{c}\right)_{n+1} = \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - \left(\tfrac{a}{c}\right)_n^2}\right)}$$

So our final formula for θ is:

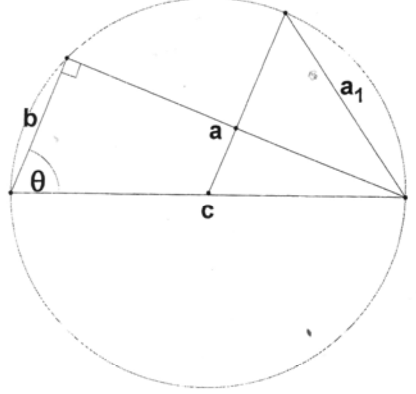
$$\theta = \tfrac{a}{c}{}_n\, 2^n$$

To fill in the last few details:

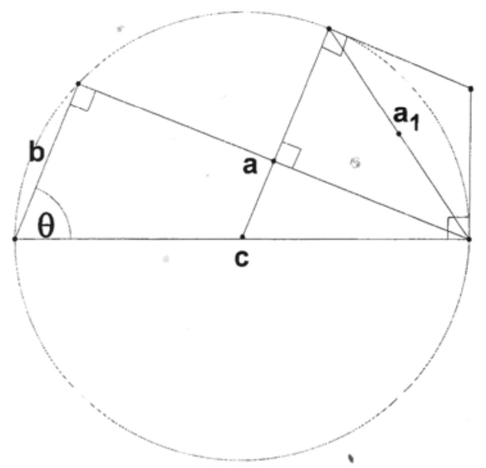
Since this algorithm is based on *the square* of **a/c**, its output can only give positive answers. Thus it gives +/-sin(x). Also, because this algorithm replicates the arc formed by theta with straight line segments *inside* the actual arc, the sum of the lines will be slightly smaller than the arc and thus less than the actual angle, making it a *lower limit* (meaning it gets bigger and bigger but always slightly smaller than the desired result.)

The Upper Limit

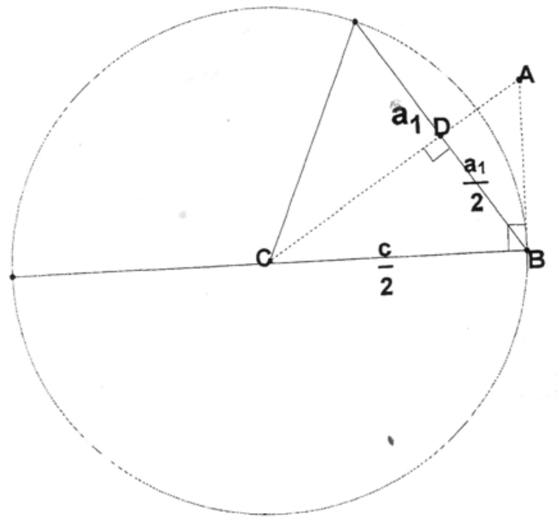
Next we will consider the upper limit and sin, which is based on this algorithm. If you refer to the earlier discussion of the lower limit, the foregoing algorithm is a lower limit because it utilizes line segments, $\tfrac{a}{c}{}_n$, that always appear *inside* the circle. So what can we do to this algorithm to make it into an upper limit? First, let's refer back to the illustration, with a_1 as the chord.



Instead of using the length of the inner chord a_1, what if we could use the length of new segments, tangent to the outside of the circle (fig 7)?



As shown above, these tangent segments would be equal in length, and would intersect directly above the midpoint of their corresponding chord (in this case a_1.) *Note: these new segments as described above are completely artificial, so their properties are determined by whoever imagines them. I gave them the above properties in hopes that geometric symmetry would give way to a simple mathematical description of them (symmetry has been known to lead to simplifications that might not otherwise occur.)*



Our goal is to find **AB**. As seen above, **DB** bisects the hypotenuse of Δ**ABC**, thus giving access to proportionality between the right triangles created inside Δ**ABC** (Δ**ABD** and Δ**CB.)**. Given that:

$$\frac{\overline{AB}}{\overline{DB}} = \frac{\overline{CB}}{\overline{CD}}$$

$$\sqrt{\overline{CB}^2 - \overline{DB}^2} = \overline{CD}$$

$$\overline{CB} = \frac{c}{2}$$

$$\overline{DB} = \frac{a_1}{2}$$

The foregoing premises provide a two variable system as follows.

$$\frac{2\overline{AB}}{a_1} = \frac{c}{2\overline{CD}}$$

$$\overline{CD} = \sqrt{\left(\tfrac{c}{2}\right)^2 - \left(\tfrac{a_1}{2}\right)^2} = \frac{\sqrt{c^2 - a_1{}^2}}{2}$$

We solve the system by substituting **CD**.

$$\frac{2\overline{AB}}{a_1} = \frac{c}{2\frac{\sqrt{c^2 - a_1{}^2}}{2}}$$

This produces the formula:

$$\overline{AB} = \frac{a_1 c}{2\sqrt{c^2 - a_1{}^2}}$$

Referring back to the conversion used earlier to measure chords in terms of **c**, all variables a_1 will become $\frac{a_1}{c}$ and all variables **c** will become 1. So our final conversion formula from inner chord to outer tangent segment is:

$$\overline{AB}_n = \frac{\frac{a}{c}_n}{2\sqrt{1 - \frac{a}{c}_n{}^2}}$$

Referring back to Figure 7, we see that it takes two tangent segments $\overline{AB}_n$ to span across the arc corresponding to their parent chord a_n. So when we multiply **AB** by 2, we derive an upper limit version of the a_1 approximation. From there, we can treat the new 2**AB** as a_1 and multiply it by 2^n to create an upper limit of the arc $\widehat{a}$.

The final upper limit formula is:

$$\theta \leq 2^n \frac{\frac{a}{c}_n}{\sqrt{1 - \frac{a}{c}_n{}^2}}$$

In this upper limit formula, a_n is found with the same iteration process as in the lower limit.

Sin

Finally, we shall explore how to approximate the sin function with the above algorithms. There are not any geometric interpretations for this next section, only analogies. To find an algorithm for sin, you must find the inverse of the algorithm for inverse sin. In this next section we shall look at how to create new algorithms by finding the inverses of the above algorithms.

We will begin with the lower limit algorithm:

$$\theta \geq 2^n \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - \left(\tfrac{a}{c}\right)_{n-1}^2}\right)}$$

$$f(x) = \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - x^2}\right)}$$

We use $\frac{a}{c}_{n-1}$ *because* $\frac{a}{c}$ is shown inside *its iteration function, supplementing an extra iteration.*

Start by finding the inverse of this function as you would any.

$$\frac{\theta}{2^n} \geq \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - \tfrac{a}{c}_{n-1}^2}\right)}$$

Knowing that the sub-n denotes “n” iterations of the function, we can compensate by iterating this by its inverse “n” times.
For example:

n=3
f(f(f(x)))

Where we have f(x) with inverse $f^{-1}(x)$ so $x = f^{-1}\left(f^{-1}\left(f^{-1}\left(f\left(f\left(f(x)\right)\right)\right)\right)\right)$.

The above function’s inverse is

$$2x\sqrt{1 - x^2}$$

So by taking the inverse of both sides of the above equation “n” times, we get

$$\tfrac{a}{c} \geq \pm f^n\left(\tfrac{\theta}{2^n}\right)$$

The algorithm can only give negative answers when x is negative so the plus/minus compensates. (Sin(x) can be negative in a positive domain or positive in a negative domain when x is between certain factors of pi.)

where

$$f(x) = 2x\sqrt{1 - x^2}$$

and “n” is *predetermined*, meaning $\dfrac{\theta}{2^n}$ remains constant. The “n” in this expression will not change as you iterate, but the number of times you actually

iterate must match the value chosen for “n.” Notice how the new algorithm

becomes an upper limit.

For the upper limit (of inverse sin):

$$|\theta| \leq \frac{2^n f^n\left(\tfrac{a}{c}\right)}{\sqrt{1 - f^n\left(\tfrac{a}{c}\right)^2}}$$

where

$$f(x) = \sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - x^2}\right)}$$

We start the same as before.

$$\frac{\theta}{2^n} \leq \frac{f^n\left(\frac{a}{c}\right)}{\sqrt{1 - f^n\left(\frac{a}{c}\right)^2}}$$

Next, we must solve for $f^n\left(\frac{a}{c}\right)$. The inverse of $f(x) = \frac{x}{\sqrt{1 - x^2}}$ is $f^{-1}(x) = \frac{x}{\sqrt{1 + x^2}}$

$$f^n\left(\tfrac{a}{c}\right) \geq \frac{\frac{\theta}{2^n}}{\sqrt{1 + \left(\frac{\theta}{2^n}\right)^2}} = \frac{\theta}{2^n\sqrt{1 + \frac{\theta^2}{2^{2n}}}} = \frac{\theta}{\sqrt{4^n + \theta^2}}$$

The final step is to put this result through the inverse of f(x) “n” times, as shown below...

$$\sin(x) \geq \pm f^n\left(\frac{\theta}{\sqrt{4^n + \theta^2}}\right)$$

Such that

$$f(x) = 2x\sqrt{1 - x^2}$$

Application

For practical purposes, these algorithms, although interesting, are usually not useful techniques for approximating the trig functions because they use square roots, which are themselves approximations. In the modern world, computer compatibility is essential for any mathematic procedure. Computers always round, so the lack of decimal places after each iteration will (as a result of chaos) will eventually cause the process to fail. For example: if we consider the iterator for inverse sin;

$$\sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - x^2}\right)}$$

We can see that x gets smaller and smaller. When x is sufficiently small, and rounded to a finite decimal, its square will be zero. Thus when is put through the algorithm, the result will be zero.

$$\sqrt{\tfrac{1}{2}\left(1 - \sqrt{1 - 0^2}\right)} = 0$$

This means that after this point, the algorithm will not work because when you multiply x_n by 2^n, you will always get zero. This is the effect of chaos on a computer. Interestingly, the sin algorithm

$$f^n(p)$$

$$f(p) = 2p\sqrt{1 - p^2}$$

$$p = \frac{x}{2^n}$$

does not experience the same effect. If we were to make **n** large enough that $\left(\frac{x}{2^n}\right)^2$ would be zero, then one iteration would produce

$$2\left(\frac{x}{2^n}\right)\sqrt{1-0} = 2\left(\frac{x}{2^n}\right) = \frac{x}{2^{n-1}}$$

After one iteration, we would have n–1 iterations to go. This would be equivalent to the nth level of approximation because we are iterating $\frac{x}{2^{n-1}}$ n–1 times. The algorithm will keep readjusting itself until you it reaches a level of approximation with which the process can take effect. This will always work unless you make **n** so big that $\frac{x}{2^n}$ is zero (instead of its square.) This makes the sin algorithm computer compatible.

Another advantage of these algorithms is their fast convergence, particularly for smaller angles and smaller ratios. This can be explained by referring back to fig 4 and fig 5. As the angle gets smaller, so do the chords used to approximate it. This means that when the angle is zero, the length of the chords used to approximate it to any level of accuracy will also be zero (as seen above when x^2 in the inverse sin algorithm was approximately zero.) The significance of this is that, at zero, the algorithms will converge infinitely fast. For example: the Taylor Series, a common approximation for the trig functions, takes dozens of terms to achieve only a few decimal places of accuracy. By contrast, the sin algorithm can achieve three correct decimal places within 10 iterations for any angle less than 2pi (the larger the angle, the slower the convergence.) For these reason, the sin algorithm could be a useful computer approximation for the trig functions.

Conclusion

Overall, despite apparent usefulness, other algorithms, such as CORDIC, specifically designed for computer efficiency far out match this one. As it turns out, the algorithms presented in this article are really geometric manifestations of algebraic properties of the trig functions; particularly the property that as x approaches 0, so do sin(x) and inverse sin(x). Using this property, a value can be repeatedly made smaller, and then renormalized as the inverse sin or sin of the value. This will become clearer in this final section.

First, we must review the first algorithm:

$$\frac{a}{c}_n = \sqrt{\frac{1}{2}\left(1-\sqrt{1-\frac{a}{c}_n{}^2}\right)}$$

When examining the effects of iteration, it becomes apparent that the outermost square root, and the innermost square, cancel out in every case except the first iteration, in which the innermost square does not cancel, and the outermost, in which the square root does not cancel (shown below.)

$$\tfrac{a}{c}{}_2 = \sqrt{\frac{1}{2}\left(1-\sqrt{1-\sqrt{\tfrac{1}{2}\left(1-\sqrt{1-\sqrt{\tfrac{1}{2}\left(1-\sqrt{1-\tfrac{a}{c}{}_0{}^2}\right)}^{\,2}}\right)}^{\,2}}\right)}$$

$$= \sqrt{\frac{1}{2}\left(1-\sqrt{1-\left(\tfrac{1}{2}\left(1-\sqrt{1-\left(\tfrac{1}{2}\left(1-\sqrt{1-\tfrac{a}{c}{}_0{}^2}\right)\right)}\right)\right)}\right)}$$

Thus,

$$\frac{a}{c}{}_n{}^2 = f^n\left(\frac{a}{c}{}_0{}^2\right)$$

$$f(x) = \tfrac{1}{2}\left(1-\sqrt{1-x}\right) \quad [1]$$

Also, the original algorithm approximates inverse sin(x) by finding θ in terms of $\frac{a}{c}$, so inverse cosine would therefore be θ in terms of $\frac{b}{c}$. To find inverse cosine, we simply start with $\frac{b}{c}{}_0$ instead of $\frac{a}{c}{}_0$, and modify the existing algorithm to convert each $\frac{b}{c}{}_n$ to an $\frac{a}{c}{}_n$. By virtue of the Pythagorean Theorem, that requires the substitution of $\sqrt{1-\frac{b}{c}{}_n{}^2}$ for every $\frac{b}{c}{}_n$. The iterator for inverse cosine is

$$\frac{b}{c}{}_{n+1} = \sqrt{\tfrac{1}{2}\left(1-\sqrt{1-\left(\sqrt{1-\tfrac{b}{c}{}_n{}^2}\right)^2}\right)} = \sqrt{\tfrac{1}{2}\left(1-\tfrac{b}{c}{}_n\right)}$$

whose inverse is

$$f^{-1}(x) = 1-2x^2 \quad [2]$$

Now things get interesting. The inverse of [1] is

$$f^{-1}(x) = 4x(1-x),$$

which can be used to approximate sin(x) in the same manor as the previous sin algorithm. This function may look familiar as the logistic function (used to model population growth through iteration.)

$$f(x) = rx(1-x)$$

with r=4. There are several interesting properties about this particular logistic function.

First, r=4 is known as the “threshold of chaos”; meaning that if r is greater than 4, the numbers produced through iteration are not bound between 0 and 1 (hence *chaos.*) In 2002, Wolfram found that the nth iterate of this function is

$$x_n = \tfrac{1}{2}\left(1-\cos\left(2^n\cos^{-1}\left(1-2x_0\right)\right)\right).$$

Here, the correlation to trigonometry is apparent, but there is another branch of mathematics this equation is linked to that has much more meaningful implications to both the original algorithm and [2].

In rational trigonometry, which deals with the squares of the trig functions to avoid transcendentals. Spread Polynomials (denoted by $S_n(s)$) are defined as follows:

$$\sin^2(n\theta) = S_n(\sin^2(\theta))$$

Similarly, Chebyshev Polynomials of the first kind, $T_n(x)$, and second kind, $U_n(x)$, are defined as

$$\cos(n\theta) = T_n(\cos(\theta))$$

Chebyshev Polynomials of the first kind can be defined as

$$T_n(x) = \cos(n \arccos(x))$$

Which is nearly identical to Mathematica's explicit formula for 4th degree logistic recursion.

$$S_2(s) = 4s(1-s) \text{ [3]}$$

$$S_m(S_n(s)) = S_{mn}(s) \text{ [4]}$$

$$T_1(x) = x \text{ [5]}$$

$$2T_n(x)^2 - 1 = T_{2n}(x) \text{ [6]}$$

Now there is a clear connection between Chebyshev Polynomials of the first kind, Spread Polynomials, and the algorithms described in this article. As seen above, iterating a second degree Spread Polynomial will achieve the same results as iterating a forth degree logistic function. The same can be said for second degree Chebyshev Polynomials of the first kind and [2]. This shows that when sine and cosine are approximated via the [1] and [2], the algorithm is really making θ very small (by repeatedly dividing it by 2), then treating the result as though it were really $\sin(\frac{\theta}{2^n})$ or $\cos(\frac{\theta}{2^n})$ (depending or what function is being approximated), and retrieving $\sin(\theta)$ or $\cos(\theta)$ by iterating a second degree Chebyshev of Spread Polynomial respectively and performing the proper renormalization (taking the square-root for Spread Polynomials.)

These methods can both be further delineated using Euler's Formula to derive a closed form version of the Chebyshev and Spread Polynomials.

$$T_n(x) = \frac{\left(x+\sqrt{x^2-1}\right)^n + \left(x+\sqrt{x^2-1}\right)^{-n}}{2}$$

$$S_n(x) = \frac{\left(x+\sqrt{x^2-1}\right)^n - \left(x+\sqrt{x^2-1}\right)^{-n}}{2i}$$

These two equations summarize all the interpretations of the algorithms used to approximate sine and cosine as applications of the fundamental rule as x approaches zero, sin(x) approaches zero.