LIBERTY UNIVERSITY

NEXUS: The Liberty Journal of Interdisciplinary Studies

Volume 1
Issue 2 *Spring 2024*

Article 2

June 2024

Circling the Square: Computing Radical Two

Isaiah Mellace
Liberty University

Joshua Kroeker
Liberty University

Circling the Square: Computing Radical Two

Isaiah Mellace, Joshua Kroeker

Abstract

Discoveries of equations for irrational numbers are not new. From Newton's Method to Taylor Series, there are many ways to calculate the square root of two to arbitrary precision. The following method is similar in this way, but it is also a fascinating derivation from geometry that has applications to other irrationals. Additionally, the equation derived has some properties that may lead to fast computation. The first part of this paper is dedicated to deriving the equation, and the second is focused on computer science implementations and optimizations.

Derivation

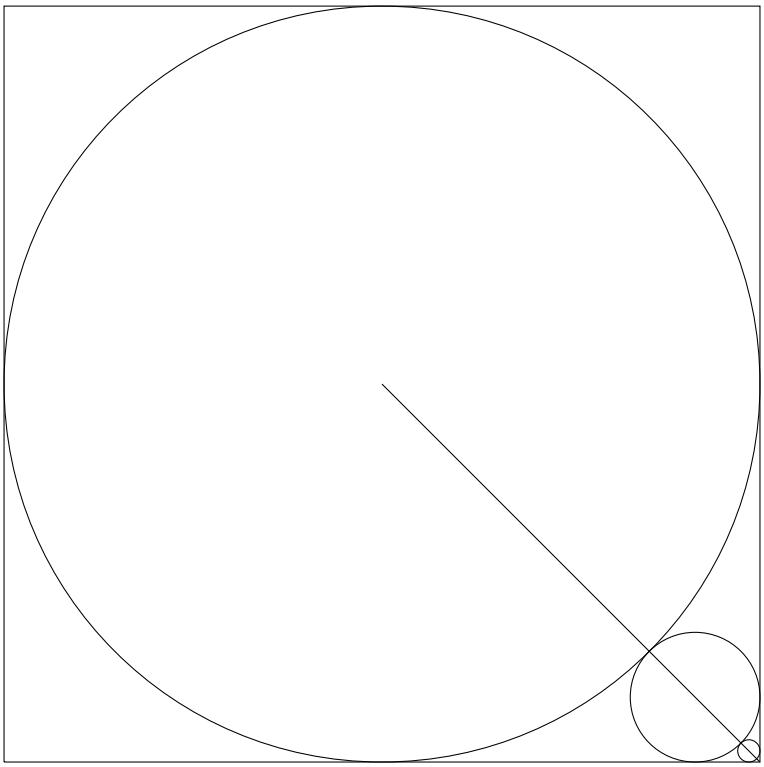


Figure 1

Beginnings

Upon investigating interesting geometric structures, we discovered **Figure 1**. This produced the following result:

$$\sqrt{2} = \lim_{n\to\infty} \frac{-3 + 6\sum_{j=0}^{n}\sum_{i=2j}^{2n+1}\binom{i}{2j}8^j 3^{i-2j}}{3 + 4\sum_{j=0}^{n}\sum_{i=2j+1}^{2n+1}\binom{i}{2j+1}8^j 3^{i-2j}}$$

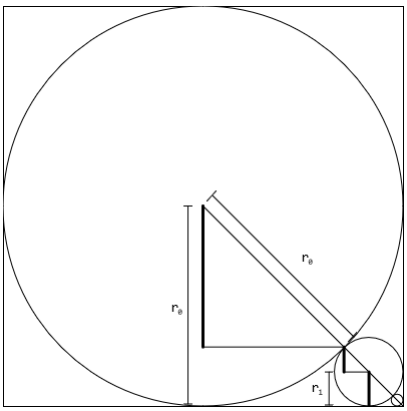
We began by observing that a square with sides of length 2 has a diagonal of $2\sqrt{2}$, thus one corner's distance is $\sqrt{2}$. We noted the original circle's radius as 1 and then added adjacent circles to approach the corner, thereby creating a sequence of radii. This led us to the equation

$$\sqrt{2} \approx r_0 + 2(r_1 + r_2 + ... + r_n) \tag{1}$$

representing the sum of the original circle's radius and twice the sum of radii of subsequent circles.

Calculating the Radii

To calculate the individual radii, we observed that we could add up the compenents of the unkown circle with those of the known circle to equal the original circle's radius:(Note: $u = \frac{1}{\sqrt{2}}$)



Thus, for r_1,

$$r_0 = u \cdot r_0 + u \cdot r_1 + r_1$$

$$r_1 = r_0\left(\frac{1-u}{1+u}\right)$$

By using the same method, we find

$$r_2 = \frac{r_0 - u \cdot r_0 - 2u \cdot r_1}{1+u}$$

$$r_2 = \frac{r_1(1+u) - 2u \cdot r_1}{1+u}$$

$$r_2 = \frac{r_1(1-u)}{1+u}$$

Which in general is

$$r_n = r_{n-1}\left(\frac{1-u}{1+u}\right)$$

Or more usefully is

$$r_n = r_0\left(\frac{1-u}{1+u}\right)^n$$

Solving for the Square Root

Now that we have the values of the radii, we use equation **(1)**

$$\sqrt{2} \approx r_0 + 2r_1 + 2r_2 + \ldots + 2r_n$$

Which is now

$$\sqrt{2} \approx r_0 + 2r_0\left(\frac{1-u}{1+u}\right) + 2r_0\left(\frac{1-u}{1+u}\right)^2 + \ldots + 2r_0\left(\frac{1-u}{1+u}\right)^n$$

$$\sqrt{2} \approx r_0(1 + 2\sum_{k=1}^{n}\left(\frac{1-u}{1+u}\right)^k)$$

$$\sqrt{2} = \lim_{n\to\infty} 1 + 2\sum_{k=1}^{n}(3-2\sqrt{2})^k \tag{2}$$

Pascal's Triangle and Factoring

At this point, we have an equation for $\sqrt{2}$. However, this is a circular definition. To remedy this, we note that the only issues come when $-2\sqrt{2}$ has an odd power, so we expand the equation and represent 3 with a, $-2\sqrt{2}$ with b, and we bold the troublesome terms. Thus,

$$(a+b) \Rightarrow a + \mathbf{b}$$
$$(a+b)^2 \Rightarrow a^2 + 2a\mathbf{b} + b^2$$
$$(a+b)^3 \Rightarrow a^3 + 3a^2\mathbf{b} + 3ab^2 + \mathbf{b}^3$$
$$\ldots$$

We generalize this through Pascal's Triangle

1
1 **1**
1 **2** 1
1 **3** 3 **1**
1 **4** 6 **4** 1
1 **5** 10 **10** 5 **1**
1 **6** 15 **20** 15 **6** 1
1 **7** 21 **35** 35 **21** 7 **1**
1 **8** 28 **56** 70 **56** 28 **8** 1
1 **9** 36 **84** 126 **126** 84 **36** 9 **1**

We then group these diagonals with functions $f(n)$ and $g(n)$ in which the latter represents the times that b has an odd power:

$$f(n) = \sum_{j=0}^{n} \sum_{i=2j}^{2n+1} \binom{i}{2j} b^{2j} a^{i-2j} \tag{3}$$

$$g(n) = \sum_{j=0}^{n} \sum_{i=2j+1}^{2n+1} \binom{i}{2j+1} b^{2j+1} a^{i-(2j+1)} \tag{4}$$

The utility of this is that if we split the summation from **(1)** into $f(n)$ and $g(n)$, we can say:

$$\sqrt{2} = \lim_{n\to\infty} 1 + 2f(n) + 2g(n)$$

$$\lim_{n\to\infty} \sqrt{2}(1 - \frac{2g(n)}{\sqrt{2}}) = \lim_{n\to\infty} 1 + 2f(n)$$

$$\sqrt{2} = \lim_{n\to\infty} \frac{1 + 2f(n)}{1 - \frac{2g(n)}{\sqrt{2}}} \tag{5}$$

Final Result

Putting it altogether, we use **(3)**, **(4)**, and **(5)** and substitute $a = 3$ and $b = -2\sqrt{2}$ to arrive at the following: (Note: 1 will be subtracted from $f(n)$ because we are not counting the case where $(a+b)^0$)

$$\sqrt{2} = \lim_{n\to\infty} \frac{1 + 2(\sum_{j=0}^{n} \sum_{i=2j}^{2n+1} \binom{i}{2j} (-2\sqrt{2})^{2j} (3)^{i-2j})}{1 - \frac{2(\sum_{j=0}^{n} \sum_{i=2j+1}^{2n+1} \binom{i}{2j+1} (-2\sqrt{2})^{2j+1} (3)^{i-(2j+1)})}{\sqrt{2}}}$$

$$\sqrt{2} = \lim_{n\to\infty} \frac{1 + 2(\sum_{j=0}^{n} \sum_{i=2j}^{2n+1} \binom{i}{2j} 8^{j} (3)^{i-2j})}{1 + 4\sum_{j=0}^{n} \sum_{i=2j+1}^{2n+1} \binom{i}{2j+1} 8^{j} (3)^{i-2j-1)}}$$

$$\sqrt{2} = \lim_{n\to\infty} \frac{-3 + 6\sum_{j=0}^{n} \sum_{i=2j}^{2n+1} \binom{i}{2j} 8^{j} 3^{i-2j}}{3 + 4\sum_{j=0}^{n} \sum_{i=2j+1}^{2n+1} \binom{i}{2j+1} 8^{j} 3^{i-2j}}$$

Implementation

While the equation derived from the previous result is correct, computing it in such a form is ineffecient. As such, the following will be a description of how this derivation can be better adapted to a computer program. (Note: While C++ is the obvious choice when it comes to computation, we have chosen to build the program in python for proof of concept.)

Intializing

To start, we base the program on equation **(5)** where we divide Pascal's Triangle into its diagonals. The difference here is that we will go through the terms horizontally and build the triangle recursively.

```
level=[1,1]
amount=int(input("how many circles (after the unit circle)? "))
while amount>0:
  leng=len(level)
  newlevel=[1,1]
  for i in range(leng):
    newlevel.insert(-1,level[i]+level[i+1])
  level=newlevel
  amount-=1
```

As we iterate through the first array to create the triangle, we add each term to either the numerator or denominator depending on the 8's exponent parity.

```
level = [1, 1]
upper = 1
lower = 1
amount = int(input("how many circles (after the unit circle)? "))
while amount > 0:
    leng = len(level)
    newlevel = [1, 1]
    for i in range(leng):
        if i < leng - 1:
            newlevel.insert(-1, level[i] + level[i + 1])
        if (i % 2 == 0 and leng % 2 == 1) or (i % 2 == 1 and leng % 2 == 0):
           upper+= 2 * level[i]*8**((leng-i)/2)*3**i
           # if it is an instance of an even power
        else:
            lower += 4 * level[i] * 8 ** ((leng - 1 - i)/2) * 3 ** i
  # if it is an instance of an odd power
    level = newlevel
    amount -= 1
```

From here we do minor optimizations and let the function have an output.

```
level=[2,2]
upper=1
lower=1
amount=int(input("how many circles (after the unit circle)? "))
for i in range(amount):
    leng=len(level)-1
    newlevel=[2,2]
    for i in range(leng+1):
            if i<leng:
                newlevel.insert(-1,level[i]+level[i+1])
            if (i%2==0 and leng%2==0) or (i%2==1 and leng%2==1):
                upper+= level[i]*8**int((leng-i)/2)*3**i
                #if it is an instance of an even power
            else:
               lower += 2 * level[i] * 8 ** int((leng - 1 - i)/2) * 3 ** i
                #if it is an instance of an odd power
    level=newlevel
newval=upper/lower
print('Estimated value of Square root of two: '+str(newval))
```

Now, we can find the average amount of decimal places gained from each iteration by dividing the error of two iterations and comparing them.

$$D(E_0, E_1) = \log_{10} \frac{E_0}{E_1}$$

This seems to approach 1.531.

The significance of this result is that if the error is compared with the amount of time it takes to compute (which was measured around 1.66 seconds for 1000 iterations) we find that (assuming constant speed of computation) it would take less than two weeks to compute a fraction that is one billion digits precise.

As the function is iterated through, we find that it is wrong to assume constant speed. The size of the array looped through increases, thereby increasing the necessary calculations. This is evidenced by 2000 iterations taking around 15.07 seconds.

Optimizations

One of the main issues with this algorithm is that it repeates calculations. For example, 8^2 and $(5)3^1 8^2$ both use the same power of 8. This could be resloved by computing all the powers of 3 and 8, storing them, and then calling them.

```
level=[2,2]
upper=1
lower=1
amount=int(input("how many circles (after the unit circle)? "))
powofeight=[8**i for i in range(amount+1)]
powofthree=[3**i for i in range(amount+1)]
for i in range(amount):
    leng=len(level)-1
    newlevel=[2,2]
    for j in range(leng+1):
```

```
            if j<leng:
                newlevel.insert(-1,level[j]+level[j+1])
            if (j%2==0 and leng%2==0) or (j%2==1 and leng%2==1):
                upper+=level[j]*powofeight[int((leng-j)/2)]*powofthree[j]
                #if it is an instance of an even power
            else:
              lower += 2* level[j] * powofeight[int((leng - 1 - j)/2)] * powofthree[
    j]
                #if it is an instance of an odd power
    level=newlevel
newval=upper/lower
print('Estimated value of Square root of two: '+str(newval))
```

This does increase the speed of the program- 2000 iterations now take about 10.86 seconds. However, this also massively increases how much data must be stored. Now, in addition to the triangle itself and the very large numerator and denominator, two giant arrays must be remembered.

There is a possibility to reduce this to a single giant array through bit manipulation. Bit manipulation uses the fact that numbers are stored in binary formats to multiply numbers by two, thereby getting rid of the need for the 8^x calculation.

```
level = [2, 2]
upper = 1
lower = 1
amount = int(input("how many circles (after the unit circle)? "))
powofthree = [3 ** i for i in range(amount + 1)]
for i in range(amount):
    leng = len(level) - 1
    newlevel = [2, 2]
    for j in range(leng + 1):
        if j < leng:
            newlevel.insert(-1, level[j] + level[j + 1])
        if (j % 2 == 0 and leng % 2 == 0) or (j % 2 == 1 and leng % 2 == 1):
            upper += (level[j] * powofthree[j]) << int(3 * (leng - j) / 2)
            #if it is an instance of an even power
        else:
            lower += (level[j] * powofthree[j]) << int((3 * (leng - j) - 1) / 2)
            #if it is an instance of an odd power
    level = newlevel
print('Numerator: ' + str(upper))
print('Denominator: ' + str(lower)))
```

This version turns out to be significantly better; on top of not needing to store an entire array, the function only took around 5.76 seconds to run 2000 iterations. In addition to this though, there are other optimizations that are possible, such as leveraging the symmetry of Pascal's Triangle to only iterate through half of the array.

Conclusion

In summary, the square root of two can be determined by leveraging concepts from circle geometry and by integrating principles from computer science. Through the combination of these two fields, we efficiently find accurate values of this irrational. In doing so, we not only uncover mathematical interconnections but also demonstrate the power of interdisciplinary approaches in solving complex problems.