POLYTECHNIC UNIVERSITY
Department of Computer and Information Science

Another Algorithm for Computing π Attributable to Archimedes: Avoiding Cancellation Errors

K. Ming Leung

Abstract: We illustrate how cancellation error can limit the accuracy that can be achieved by an algorithm for computing π, and how that error can be avoided.

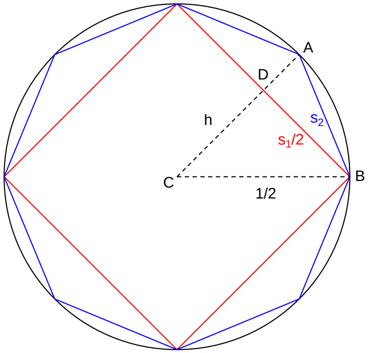
Directory

- **Table of Contents**
- **Begin Article**

In 250 B.C., the Greek mathematician Archimedes estimated the value of π by approximating the circumference of a circle by the perimeter of a regular polygon having $2n$ sides, where $n = 2, 3, \ldots$. He knew that the approximation becomes better and better as n increases. In addition he was able to derive a recursion formula relating the perimeter of a regular polygon to the perimeter of another regular polygon having twice as many sides. One version of his method is described here.

He considered a circle with diameter equal to 1, hence its circumference is π. Inside the circle he inscribed a square. The perimeter of the square is less than the circumference of the circle, and so it is a lower bound for π. Archimedes then considered an inscribed octagon, hexadecagon, etc., each time doubling the number of sides of the inscribed polygon, and producing better and better estimates for π.

Let p_n be the perimeter of the inscribed polygon with 2^{n+1} sides, where $n = 1, 2, \ldots$. Let the length of each side be denoted by s_n. Then p_1 is the perimeter of the inscribed square, and from the geometry we know that $s_1 = 1/\sqrt{2}$ and so $p_1 = 4s_1 = 2\sqrt{2}$. The perimeter of an


A
D
h
s_2
$s_1/2$
C
B
1/2

inscribed octagon p_2 is equal to $8s_2$, and the perimeter of an inscribed hexadecagon p_3 is equal to $16s_2$. In general we see that

$$p_n = 2^{n+1} s_n \qquad \text{for} \qquad n = 1, 2, \ldots. \tag{1}$$

In the limit that n goes to infinity, the regular polygon becomes more and more like the circle and so we have $p_\infty = \pi$. So far the only perimeter we know is p_1 for the square. However Archimedes was able to obtain a recursive formula for the perimeters of the polygons. This allowed him to compute p_2 from p_1, and p_3 from p_2, and so on.

To obtain such a recursive formula, let us see how to compute p_2 from p_1 using geometry only. Since triangle BCD is a right-angled triangle, Pythagoras theorem gives

$$h = \sqrt{\left(\frac{1}{2}\right)^2 - \left(\frac{s_1}{2}\right)^2}. \tag{2}$$

Actually it is clear that $h = s_1/2$, however this geometric relation does not apply to other polygons with a different n. Since we want a recursive relation that applies to all n, that relation must not be used. The length AC is $1/2$ and so the length AD is $1/2 - h$. Applying

Pythagoras theorem to triangle ABD one has

$$s_2 = \sqrt{\left(\frac{s_1}{2}\right)^2 + \left(\frac{1}{2} - h\right)^2}$$

Substituting h from Eq. (2) into this equation, expanding and then simplifying the expression yields the result

$$s_2 = \sqrt{\frac{1}{2} - \sqrt{\left(\frac{1}{2}\right)^2 - \left(\frac{s_1}{2}\right)^2}}.$$

A careful inspection of the diagram reveals that such a relation holds between s_3 and s_2, etc. Thus in general one has the recursive relation

$$s_{n+1} = \sqrt{\frac{1}{2} - \sqrt{\left(\frac{1}{2}\right)^2 - \left(\frac{s_n}{2}\right)^2}}, \tag{3}$$

for $n = 1, 2, \ldots$. We then obtain a recursive relation involving the perimeters using Eq. (1). The result is

$$p_{n+1} = 2^{n+1}\sqrt{2\left(1-\sqrt{1-\left(\frac{p_n}{2^{n+1}}\right)^2}\right)}. \tag{4}$$

Starting the iteration with $p_1 = 2\sqrt{2}$ and $n = 1$ we then get p_2, etc. One can easily write a program to implement that. The result computed using MATLAB (in double-precision) is shown below. Notice that because of the iterative nature of the algorithm, the program cannot be vectorized. The computed values for p_1 to p_{31} are shown below.

```
1    2.82842712474619
2    3.06146745892072
3    3.12144515225805
4    3.13654849054594
5    3.14033115695474
6    3.14127725093276
7    3.14151380114415
8    3.14157294036788
9    3.14158772527996
```

```
10    3.14159142150464
11    3.14159234561108
12    3.14159257654500
13    3.14159263346325
14    3.14159265480759
15    3.14159264532122
16    3.14159260737572
17    3.14159291093967
18    3.14159412519519
19    3.14159655370482
20    3.14159655370482
21    3.14167426502176
22    3.14182968188920
23    3.14245127249413
24    3.14245127249413
25    3.16227766016838
26    3.16227766016838
27    3.46410161513775
28    4
29    0
```

```
  30    0
  31    0
```

Notice that the values at first increase (as expected) and seem to converge to the correct value for π until p_{14}. Notice that this value is slightly larger than π, but how can an inscribed polygon have a perimeter larger than the inscribing circle! After that the values decrease slightly (again this is rather alarming since we do not expect the perimeters to ever decrease with increasing n.) and gradually increase above π and drift further and further away from π. After the 28th term the computed perimeters all become zero.

The reason for that is obvious from Eq. (4). Since p_n is expected to be around 3, the factor $p_n/2^{n+1}$, which is actually the length of each side of the polygon, decreases as we iterate. As soon as

$$\left(\frac{p_n}{2^{n+1}}\right)^2 \leq \frac{\epsilon_{\rm mach}}{2},$$

we have

$$1 - \left(\frac{p_n}{2^{n+1}}\right)^2 \to 1, \quad \text{and so} \quad p_{n+1} \to 0,$$

due to cancellation in the above iteration formula in Eq. (4).

To estimate the value of n at which that happens, we set $p_n \approx 3$ in the equation

$$\left(\frac{p_n}{2^{n+1}}\right)^2 = \frac{\epsilon_{\text{mach}}}{2},$$

and solve for n to get

$$n = \log_2\left(p_n\sqrt{\frac{2}{\epsilon_{\text{mach}}}}\right) \approx \left(3\sqrt{\frac{2}{1.1\times 10^{-16}}}\right) \approx 29.$$

This agrees with the results obtained from the program.

Before we deal with the problem associated with cancellation , we want to address some issues related to numerical efficiency. First, notice from Eq. (4) that on the right-hand side p_n is divided by 2^{n+1} but p_{n+1} is obtained by multiplying a certain factor with 2^{n+1}. Clearly we can avoid all that if we perform the iteration on s_n instead of p_n, *i.e.* we iterate using equation Eq. (3). The cancellation problem is still present since as n increases the length of each segment of the polygon s_n decreases.

Second, notice that s_n is being squared but to obtain s_{n+1} we need to take a square root. Clearly all that can be avoided if we work with a new variable $t_n = s_n^2$. Changing to this new variable, we take the square of Eq. (3) to obtain

$$t_{n+1} = \frac{1}{2}\left(1 - \sqrt{1 - t_n}\right).$$

We can iterate using this formula starting with $t_1 = s_1^2 = 1/2$. The perimeters are then given by

$$p_n = 2^{n+1} s_n = 2^{n+1}\sqrt{t_n}.$$

Avoiding the squaring and the taking of a square root actually saves about 40% of the computing time. Of course the cancellation problem is still not yet solved.

Now we solve the cancellation problem. The key is to realize that the above equation looks almost like the quadratic formula. The cure is of course exactly the same. So we write

$$t_{n+1} = \frac{1}{2}\left(1 - \sqrt{1 - t_n}\right)\left(\frac{1 + \sqrt{1 - t_n}}{1 + \sqrt{1 - t_n}}\right) \tag{5}$$

$$= \frac{1}{2}\frac{1 - (1 - t_n)}{1 + \sqrt{1 - t_n}} = \frac{1}{2}\frac{t_n}{1 + \sqrt{1 - t_n}}. \tag{6}$$

Now this recursive formula does not have any cancellation problem. When t_n is small, the denominator simply becomes 2.

Let us do one more thing to see if we can further improve the efficiency of the above algorithm by a simple re-scaling of the variable. Let us introduce a scaled variable g_n by writing $t_n = \alpha g_n$. One can see that Eq. (5) becomes

$$g_{n+1} = \frac{g_n}{2 + 2\sqrt{\alpha}\sqrt{\frac{1}{\alpha} - g_n}}.$$

Therefore if we let $\alpha = 1/4$ so that $2\sqrt{\alpha} = 1$, the iterative relation for g_n becomes

$$g_{n+1} = \frac{g_n}{2 + \sqrt{4 - g_n}}. \tag{7}$$

There is one fewer multiplication in this formula than the one in Eq. (5). Notice that $t_n = g_n/4 = s_n^2$, and $p_n = 2^n\sqrt{g_n}$. Thus we start the iteration with $g_1 = 4s_1^2 = 2$. The computed values for p_1 to p_{31} are shown below.

```
 1    2.828427124746190
 2    3.061467458920718
 3    3.121445152258052
 4    3.136548490545939
 5    3.140331156954753
 6    3.141277250932773
 7    3.141513801144301
 8    3.141572940367091
 9    3.141587725277160
10    3.141591421511200
11    3.141592345570118
12    3.141592576584873
13    3.141592634338563
14    3.141592648776986
15    3.141592652386591
16    3.141592653288993
17    3.141592653514593
18    3.141592653570993
19    3.141592653585093
20    3.141592653588618
```

```
21      3.141592653589500
22      3.141592653589720
23      3.141592653589775
24      3.141592653589789
25      3.141592653589793
26      3.141592653589794
27      3.141592653589794
28      3.141592653589794
29      3.141592653589794
30      3.141592653589794
```

After 26 iterations, the result converges to a value as close to π as is possible using IEEE double-precision. The result does not change with further iterations. If one computes the relative error of this converged value, the result is `eps`. That means that we have obtained the most accurate result as we possibly can on this floating point system.