Comparing the speed of two methods of approximating the value of π — a computational approach

To what extent can a method of approximation of the value π be computationally more efficient than another?

Word count:

Contents

1 Introduction

The value of π has been researched for many years, although under different names, and the amount of different approaches to reach the value is large. The value has been found through many processes, be it analytically, the most popular, geometrically, or through more obscure or convoluted methods, such as the possibility to approximate the value using physics akin to those from a simple game of billiards. [1]

This paper seeks to examine the extent at which two historical methods of approximation of the value π, namely the approaches suggested by the aforementioned mathematicians Madhava and Viète, differ in terms of computational speed and speed, and explain these differences. This paper does not however, suggest a method to use for computation but rather seeks to only compare the speed of a geometrically derived formula and a analytically derived one.

This research could prove useful in the field of computer science, as there is always a demand for faster and more efficient programs in an ever-changing society.

Furthermore, a research based on computational speed of two different kinds approaches to the constant π has not been done to date.

2 Background

2.1 Focus on two methods

In this paper, we focus on two methods for approximating the value of π. The first by French mathematician François Viète, and another discovered by Madhava of Sangamagrama and rediscovered by Swiss mathematician Leibniz. Two methods with different approaches have been chosen for comparison, the process for original discovery of these methods will be explained. The two in question are one based on the infinite series definition of an inverse trigonometric function and one where the value is derived using geometry, by mathematician Viète. These two mathematical methods were chosen as they are both represent a first occurrence in mathematics: Madhava was the first to find the infinite series expansion of the inverse tangent function and Viète was the first mathematician to use and infinite product in calculation.

2.1.1 Madhava-Leibniz method

Madhava is the first mathematician known to have found the series notation for the inverse tangent function, in the Kerala school of mathematics of Medieval India [2]. The research made at the Kerala school was documented by many mathematicias of their time, namely the astronomer-mathematician Jyesthadeva in his treatise *Yukti-Bhasa* written in the Malayam language [3]. Madhava's infinite series expansion of the inverse tangent function can also be found in this treatise in this translation, adapted from (R. C. Gupta 67-70) [4]:

The first term is the product of the given sine and radius of the desired arc divided by the cosine of the arc. The succeeding terms are obtained by a process of iteration when the first term is repeatedly multiplied by the square of the sine and divided by the square of the cosine. All the terms are then divided by the odd numbers 1, 3, 5, ... The arc is obtained by adding and subtracting respectively the terms of odd rank and those of even rank. It is laid down that the sine of the arc or that of its complement whichever is the *smaller should be taken here as the given sine. Otherwise the terms obtained by this above iteration will not tend to the vanishing magnitude.*

2.1.2 Viète's method

François Viète approached the value of π from a geometric standpoint, and found an infinite product. He was able to calculate π to a place of 9 decimal points, in the year 1593 [5], using his method. His method is reminiscent of Archimedes' method, where the length of a side is calculated [6], but differs in that it consists of finding the area of a polygon of n sides in a circle of constant radius, rather than the circumference. As the value of n is increased, the area of the n-gon tends toward the area of a circle. The geometric origin of this formula can be found using simple right-angle trigonometry, by first finding the lengths OH and subsequently BD in 2.1.2. With the radius of the circle with center $R = OB$,

$OH = R \cos \alpha$

and

$BD = 2BH = 2R \sin \alpha$

Since the equation for the area of a polygon is defined as $A = \frac{p \cdot a}{2}$, where p is the perimeter of the polygon and a is the apothem, in this case $BD \cdot n$ and OH respectively, let A_n equal the area of the polygon with n sides such that:

$A_n = \frac{OH \cdot BD \cdot n}{2}$

$A_n = \frac{R \cos \alpha \cdot 2R \sin \alpha \cdot n}{2} = nR^2 \sin \alpha \cos \alpha$

And if n is multiplied by 2, the angle $\angle\alpha$ is divided by 2, and the new area becomes:

$A_{2n} = 2nR^2 \sin \frac{\alpha}{2} \cos \frac{\alpha}{2}$

So it can be written that, by definition, the ratio of the area of an n-gon to one of a $2n$-gon is

$\frac{A_n}{A_{2n}} = \frac{nR^2 \sin \alpha \cos \alpha}{2nR^2 \sin \frac{\alpha}{2} \cos \frac{\alpha}{2}} = \frac{\sin 2\alpha}{2 \sin \alpha}$

Which through the trigonometric identity $\sin 2\theta = 2 \sin \theta \cos \theta$ can be simplified to:

$\frac{A_n}{A_{2n}} = \frac{2\sin\alpha\cos\alpha}{2\sin\alpha} = \cos\alpha$

It can be then written that, through a new variable P,

$P = \frac{A_n}{A_{2n}}\frac{A_{2n}}{A_{4n}}\frac{A_{4n}}{A_{8n}}\dots\frac{A_{(k-2)n}}{A_{kn}}\frac{A_{kn}}{A}$

where A is the area of the circle of radius R in 2.1.2.

So $P = \frac{A_n}{A}$, since the values A_{kn} cancel, and followingly, it is true that $P = \frac{A_n}{A} \Leftrightarrow A = \frac{A_n}{P}$. The value of $R = 1$ in this case, and since the area of a circle is defined by $A = \pi R^2$, therefore:

$\pi = \frac{A_n}{P}$

By definition, we can say that as the value k approaches infinity, the area of the kn-gon aproaches that of a circle, and therefore, the value of π.

$\frac{A_n}{\cos\alpha\cos\frac{\alpha}{2}\cos\frac{\alpha}{4}\dots} \rightarrow_{k\rightarrow\infty} \pi$

Where the value of A_n is the area of the first polygon, with $n = 4$ sides, and as such $A_n = 4\sin 45\cos 45 = 2$. We can define:

$U_0 = \cos a = \cos 45 = \frac{1}{\sqrt{2}}$

$U_1 = cos\frac{\alpha}{2}$

Which we can, through the trigonometric identity $\cos^2\theta = \frac{1}{2} + \frac{1}{2}\cos 2\theta$, simplify as $U_1 = \sqrt{\frac{1}{2} + \frac{1}{2}U_0}$

So it can be said that $U_n = \sqrt{\frac{1}{2} + \frac{1}{2}U_{n-1}}$, which leads to a fully defined expression for the value of pi, using an infinite product:

$\pi = \frac{2}{\prod\limits_{k=0}^{\infty} U_k}, U_0 = \frac{1}{\sqrt{2}}, U_n = \sqrt{\frac{1}{2} + \frac{1}{2}U_{n-1}}$ [1]

These expressions, when under a single expression result in the aforementioned formula with nested roots.

[1] where $\prod$ signifies a product. Similar expression to $\sum$

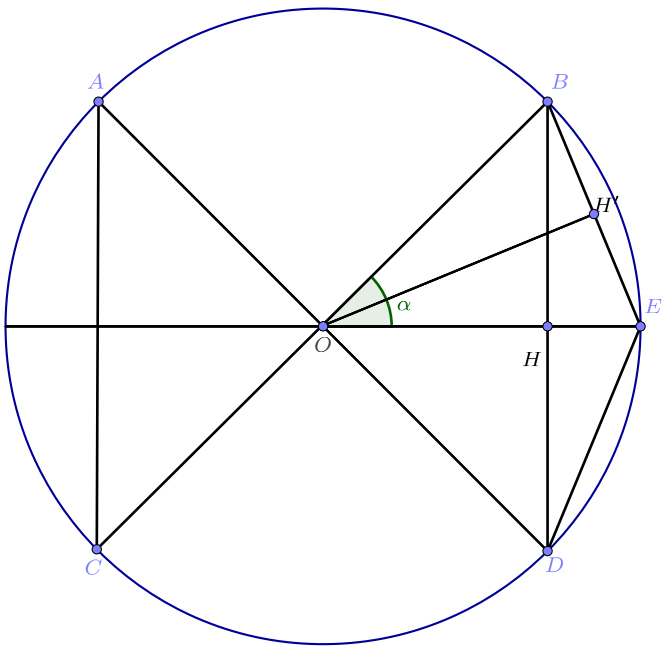


Figure 1: Circle with 1 segment from a n-gon with point H and 2 segments from an $2n$-gon, one of which on point H', inscribed in a circle of radius OB, adapted from (Boris Gourévitch) [7]

3 Computational approach

3.1 The variables

The dependent variable of this experiment is the time taken t by the program to approximate a given number n of correct decimal value of the constant π.

The value n will be altered in order to avoid possible similar convergence rates at a small amount of decimal places, and multiple trials will be run to decrease margin of error. Other variables of the experiment will be controlled. For example, the experiment will be run on a same isolated system, a virtual machine, with a minimal amount of processes running simultaneously to avoid any possible variance in results.

3.2 Implementing in Python

A Python application was programmed (see appendix) in order to run the two methods aforementioned, and manage the collection of data.

The program assigns the time before the execution of the method to a variable `t1` with the `time.time()` Python function. At each iteration of the method, the number of valid decimal places of the resultant approximation are counted and once a specified threshold is reached, a new `t2` time variable is assigned and the time taken, defined by the difference between `t2` and `t1`, is stored. This process is repeated for all specified decimal accuracies and for both methods. The times recorded were stored in a `.csv` file for further analysis.

The `mpmath` library was used for the floating point operations required for comparison between approximated values and the constant π that wouldn't have been possible using standard Python libraries [8].

4 Analysis of the results

4.1 Presentation of the data

4.1.1 Tabular presentation

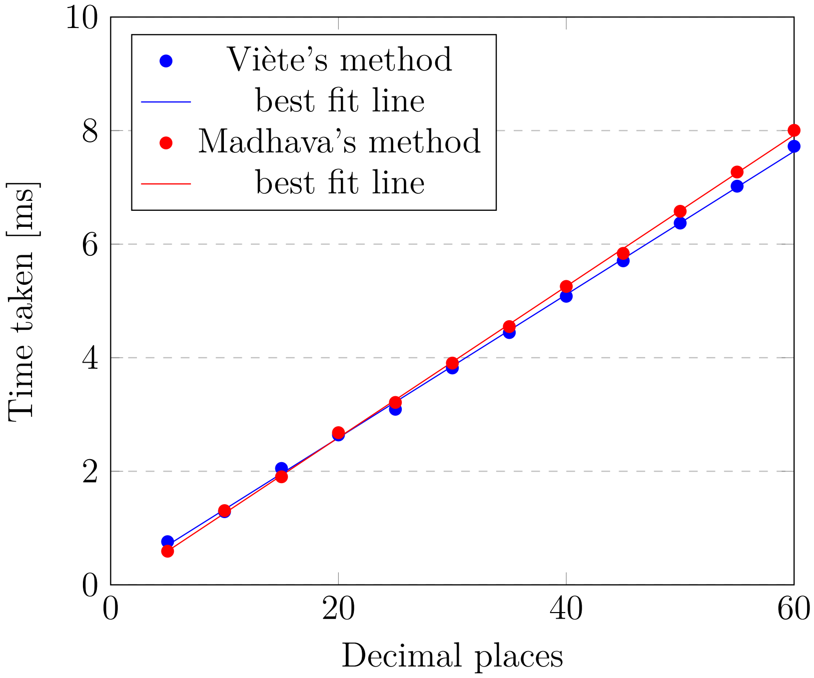
The arithmetic mean of a decimal place was caluclated for 100 trials and the data points in the table shown below were found (rounded to 3 s.f.):

Decimal place	Average time for approximation using Viete's method, in milliseconds (ms)	Average time for approximation using Madhava's method method, in milliseconds (ms)
5	0.760	0.592
10	1.29	1.31
15	2.05	1.9
20	2.64	2.68
25	3.09	3.21
30	3.82	3.9
35	4.44	4.55
40	5.08	5.26
45	5.71	5.84
50	6.37	6.58
55	7.02	7.27
60	7.72	8.00

4.1.2 Graphical presentation

To better show the trend in the data collected, the data has been presented in a chart below. The blue markers and line of best-fit respresent the results received from Viète's method while the red represent

Comparing the time taken for approximating the value of π



4.2 Analysis

5 Conclusion

Works Cited

[1] G. Galperin. "PLAYING POOL WITH π (THE NUMBER π FROM A BILLIARD POINT OF VIEW)". In: *Regular and Chaotic Dynamics* 8.4 (2003), pp. 375–393. DOI: 10.1070/rd2003v008n04abeh000252. URL: http://rcd.ics.org.ru/upload/iblock/007/RCD080402.pdf.

[2] Jonathan M. Borwein, Scott T. Chapman, and Scott T. Chapman. "I Prefer Pi: A Brief History and Anthology of Articles in the American Mathematical Monthly". In: *The American Mathematical Monthly* 122.3 (2015), pp. 198–199. ISSN: 00029890, 19300972. URL: http://www.jstor.org/stable/10.4169/amer.math.monthly.122.03.195 (visited on 01/23/2020).

[3] J J O'Connor and E F Robertson. *Madhava of Sangamagramma*. Nov. 2000. URL: https://web.archive.org/web/20060514012903/http://www-gap.dcs.st-and.ac.uk/~history/Biographies/Madhava.html.

[4] R C Gupta. *The Madhava-Gregory series*. 1973, pp. 67–70.

[5] Rick Kreminski. "π to Thousands of Digits from Vieta's Formula". In: *Mathematics Magazine* 81.3 (2008), p. 201. ISSN: 0025570X, 19300980. URL: http://www.jstor.org/stable/27643107 (visited on 01/23/2020).

[6] RICHARD LOTSPEICH. "Archimedes' Pi—an Introduction to Iteration". In: *The Mathematics Teacher* 81.3 (1988), p. 208. ISSN: 00255769. URL: http://www.jstor.org/stable/27965770.

[7] Boris Gourévitch. *François Viete*. Apr. 2013. URL: http://www.pi314.net/fr/viete.php (visited on 01/24/2020).

[8] Fredrik Johansson et al. *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 0.18)*. Dec. 2013. URL: http://mpmath.org/.

A Appendix

A.1 Python program

This application was run on Python version 3.9.1, on a virtual machine running the Debian operating system under QEMU/KVM on a Intel i5-2500 processor. Used the `mpmath` library for better floating-point precision [8].

```
import time
from mpmath import *
import csv
import pandas as pd



mp.dps = 100
PI_CONST = mp.pi



# Function that determines if the approximated value of
# pi is correct to a specified decimal
def decimal_is_correct(pi, decvalue):
    #pi_diff = str(abs(pi))
    zeros = 0
    pistr = str(pi)[2:]
    piconst = str(PI_CONST)[2:]
    for i in range(len(pistr)):
        if pistr[i] != piconst[i]: break
        else:
            zeros += 1
    if zeros == decvalue:
```

```python
            return True
        else:
            return False


# Function that approximates pi using the Madhava-Leibniz
# method
def madhavaleibniz(decimals):
    piapprox = 0
    i = 0

    t1 = time.time()

    while not decimal_is_correct(piapprox * mp.sqrt(12), decimals):
        # This is a direct mirror of the summation from the formula
        piapprox += mp.power(-3, -i) / (2*i+1)
        i += 1
    piapprox *= mp.sqrt(12)
    t2 = time.time()

    # Return the time spent (t2-t1) getting d value of decimal places
    return t2-t1


# Function that approximates pi using Viete's method
def viete(decimals):
    piapprox = 1
    numer = 0
```

```
    t1 = time.time()
    while not decimal_is_correct((1.0 / piapprox) * 2.0, decimals):
        numer = mp.sqrt(2.0 + numer)
        piapprox *= (numer / 2.0)
    piapprox = (1.0 / piapprox) * 2.0

    t2 = time.time()

    # Return the time spent (t2-t1) getting d value of decimal places
    return t2-t1



if __name__ == "__main__":
    decimals = [i for i in range(0, 65, 5)]
    trials = 100

    # Exporting data for plotting and analysis
    f = open(r'out.csv', 'w')
    fieldnames = ['decimals', 'viete', 'madhava']
    writer = csv.DictWriter(f, fieldnames=fieldnames)
    writer.writerow({'decimals':'decimals', 'viete':'viete',
    'madhava':'madhava'})

    for dec in decimals:
        for t in range(trials):
            writer.writerow({'decimals':dec, 'viete':viete(dec),
            'madhava':madhavaleibniz(dec)})
```