IB HL MATHEMATICS: Analysis and Approaches

Internal Assessment

An Analysis of Methods of Approximating π Using Iterative Computer Algorithms.

Table of Contents

1. Overview

1.1 Introduction

π is a crucial universal constant in mathematics and physics, governing the behavior of circles and appearing in various applications from trigonometry to infinite series. A notable detail regarding π is that it is irrational, and therefore its exact value is unknowable and would require an infinite number of digits to express. However, even if the exact value of π cannot be calculated, its value can be approximated. Mathematicians have long attempted to approximate π, and have constantly refined their methods throughout history. Additionally, the advent of computing technology allows computers to make calculations in a fraction of the time it would take to do by hand, allowing mathematicians to calculate π to far higher precision. This exploration will outline several methods for calculating π using computer programs and compare their efficiency in calculating π to several decimal places.

1.2 Personal Engagement

My first experience with π was "approximating" it as slightly more than 3 using a cylinder, a ruler, and a sheet of paper, as a young child. However, without mathematical knowledge or training, I was unable to carry this further. When I formally learned about π, I could not imagine how its value could be mathematically calculated. Every method I could think of appeared to be circular to me, seemingly requiring preexisting knowledge of its value as part of the derivation. I also wondered how mathematicians could know how many decimal places of π they had calculated without being able to take the exact value and compare it to their approximations. Later on, by chance, I learned a method of approximating π while reading a book: Drawing a polygon with more and more sides until it approached a circle, then using conventional geometry to calculate its perimeter, which approached the circumference of a circle. The final piece of the puzzle came in my second year of Mathematics Higher Level, when I learned about Maclaurin series. This allowed me to express trigonometric functions, which could relate π to rational numbers, in terms of rational functions. Now in possession of this knowledge, I was confident that I could approximate π myself.

1.3 Methodologies

In this exploration, several computer programs were written that approximate values close to π . This exploration did not compare calculated values with the literature value of π, as mathematicians calculating π would not have access to it. Instead, each program outputted two approximate values, one greater than π and one less than π. The number of decimal places of π approximated was then determined by comparing the two bounds. The digits before the point where the bounds diverge must be digits of π, so the number of digits until the two bounds diverge are the number of decimal places approximated.

2. Archimedes' Method

2.1 Explanation

One method for approximating π is bounding the circumference of a circle with the perimeters of regular polygons. This method was devised by Archimedes, and is therefore called the Archimedes' method. As the perimeters of polygons approach that of a circle when there are a large number of sides, drawing polygons with extremely high numbers of sides both larger than and smaller than a circle can be used to bound the circumference. Given that π is defined as the ratio between the circumference and diameter of a circle, approximating the circumference of a circle with known diameter can be used to approximate π.

The perimeter of polygons inscribed and circumscribed on a unit circle with $3 * 2^{k-1}$ sides can be calculated for any k using a recursive formula. They approximate the circumference of a circle with radius 1. Since the diameter of a circle is $2\pi r$, where r is the radius, the perimeter of the polygons will approximate 2π. Therefore, dividing the perimeter of the inscribed and the circumscribed polygons by 2 will give a lower bound and an upper bound, respectively, for π.

2.2 Proof and Derivation

Let the perimeter of the circumscribed polygon with $3 * 2^{k-1}$ sides be A_k . Let the perimeter of the inscribed polygon with $3 * 2^{k-1}$ sides be B_k .

Lemma 1: $A_k = 3 \cdot 2^k tan(\frac{120^o}{2^k})$

Lemma 2: $B_k = 3 \cdot 2^k sin(\frac{120^o}{2^k})$

Proof of Lemma 1:

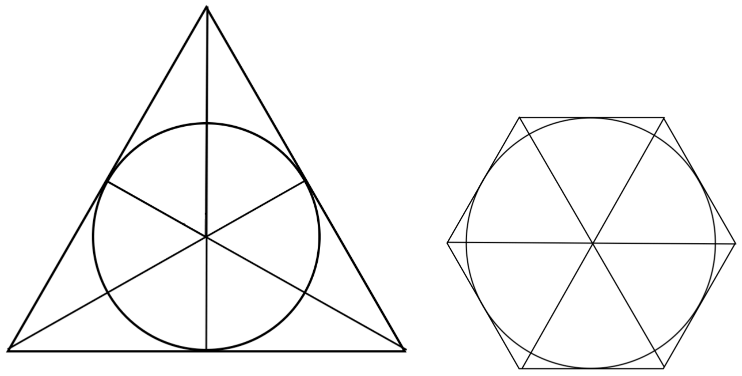


Figure 1: Circumscribed Polygons With $3 * 2^{k-1}$ Sides

Each of the vertices of the polygon can be connected to the center of the circle, creating $3 * 2^{k-1}$ isosceles triangles. Note that since the midpoint of each side of a circumscribed polygon is tangent to the circle, the heights of each of these triangles are radii. Call the angle of the vertex that touches the centre of the circle the "central angle". Also, note that the sum of the bases of these triangles is the perimeter of the polygon. Each of these triangles can then be bisected along the height, resulting in $3 * 2^{k}$ right triangles. Note that the legs of these right triangles that are adjacent to the central angle are the radii of the circle. The tangent of an angle in a right triangle is equal to the length of the opposite leg divided by the length of the adjacent leg. Thus, the leg that lies on the perimeter of the polygon is the radius of the circle multiplied by the tangent of the central angle. Since there are $3 * 2^{k}$ right triangles and the radius of the circle is 1, the perimeter of the polygon is $3 \cdot 2^{k} tan(\frac{120^{o}}{2^{k}})$

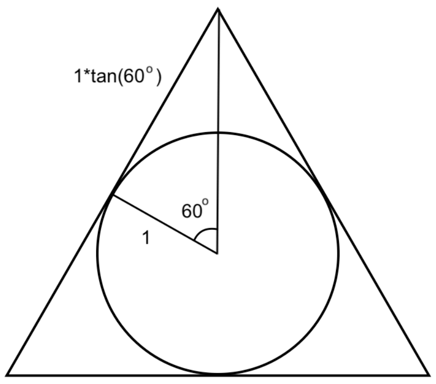


Figure 2: Right Triangle Within the Circumscribed Polygon

Proof of Lemma 2:

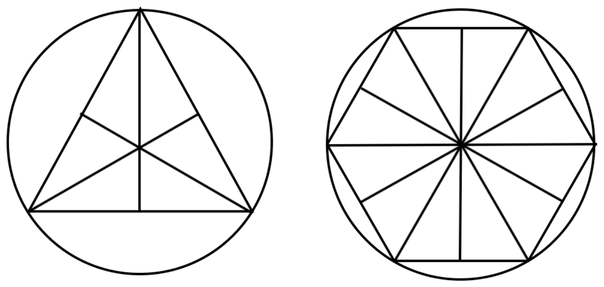


Figure 3: Inscribed Polygons With $3 * 2^{k-1}$ Sides

Each of the vertices of the polygon can be connected to the center of the circle, creating $3 * 2^{k-1}$ isosceles triangles. Note that since the vertices of the inscribed polygon lie on the circle, the two equal side lengths of these circles are the radii. Call the angle of the vertex that touches the center of the circle the "central angle". Also note that the sum of the bases of these triangles is the perimeter of the polygon. Each of these triangles can then be bisected along the height, resulting in $3 * 2^{k}$ right triangles.

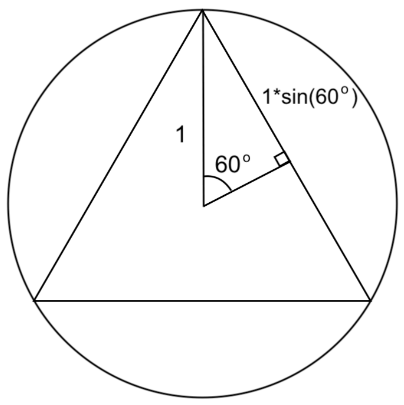


Figure 4: Right Triangle Within the Inscribed Polygon

Note that the hypotenuses of these right triangles are the radii of the circle. The sine of an angle in a right triangle is equal to the length of the opposite leg divided by the length of the hypotenuse. Thus, the leg that lies on the perimeter of the polygon is the radius of the circle multiplied by the sine of the central angle. Since there are $3 * 2^k$ right triangles and the radius of the circle is 1, the perimeter of the polygon is $3 \cdot 2^k sin(\frac{120^o}{2^k})$

Lemma 3:

$$A_{k+1} = \frac{2A_k B_k}{A_k + B_k}$$

Lemma 4:

$$B_{k+1} = \sqrt{A_{k+1} B_k}$$

Proof of Lemma 3:

$$\frac{2(3\cdot 2^{k} tan(\frac{120^{o}}{2^{k}}))(3\cdot 2^{k} sin(\frac{120^{o}}{2^{k}}))}{3\cdot 2^{k} tan(\frac{120^{o}}{2^{k}})+3\cdot 2^{k} sin(\frac{120^{o}}{2^{k}})}$$

$$= \frac{9\cdot 2^{2k+1} tan(\frac{120^{o}}{2^{k}}) sin(\frac{120^{o}}{2^{k}})}{3\cdot 2^{k}(tan(\frac{120^{o}}{2^{k}})+sin(\frac{120^{o}}{2^{k}}))}$$

$$= \frac{3\cdot 2^{k+1}\frac{sin^{2}(\frac{120^{o}}{2^{k}})}{cos(\frac{120^{o}}{2^{k}})}}{\frac{sin(\frac{120^{o}}{2^{k}})}{cos(\frac{120^{o}}{2^{k}})}+sin(\frac{120^{o}}{2^{k}})}$$

$$= \frac{3\cdot 2^{k+1}\frac{sin(\frac{120^{o}}{2^{k}})}{cos(\frac{120^{o}}{2^{k}})}}{\frac{1}{cos(\frac{120^{o}}{2^{k}})}+1}$$

$$= \frac{3\cdot 2^{k+1} sin(\frac{120^{o}}{2^{k}})}{1+cos(\frac{120^{o}}{2^{k}})}$$

$$= 3\cdot 2^{k+1}\frac{sin(\frac{120^{o}}{2^{k}})}{1+cos(\frac{120^{o}}{2^{k}})}$$

Since a tangent half angle formula is

$$tan(\frac{\theta}{2}) = \frac{sin\theta}{1+cos\theta}$$

(Haese, Humphries, Sangwin, & Vo, 2019)

The formula is thus

$$= 3\cdot 2^{k+1} tan(\frac{120^{o}}{2^{k+1}})$$

$$= A_{k+1}$$

Proof of Lemma 4:

$$\sqrt{(3 \cdot 2^{k+1} tan(\frac{120^o}{2^{k+1}}))(3 \cdot 2^{k} sin(\frac{120^o}{2^{k}}))}$$

$$= \sqrt{9 \cdot 2^{2k+1} tan(\frac{120^o}{2^{k+1}}) sin(\frac{120^o}{2^{k}})}$$

By the sine double angle formula (Haese, Humphries, Sangwin, & Vo, 2019)

$$= \sqrt{9 \cdot 2^{2k+1} tan(\frac{120^o}{2^{k+1}})(2 sin(\frac{120^o}{2^{k+1}}) cos(\frac{120^o}{2^{k+1}}))}$$

$$= \sqrt{9 \cdot 2^{2k+2} tan(\frac{120^o}{2^{k+1}}) sin(\frac{120^o}{2^{k+1}}) cos(\frac{120^o}{2^{k+1}})}$$

$$= \sqrt{9 \cdot 2^{2k+2} \frac{sin(\frac{120^o}{2^{k+1}})}{cos(\frac{120^o}{2^{k+1}})} sin(\frac{120^o}{2^{k+1}}) cos(\frac{120^o}{2^{k+1}})}$$

$$= \sqrt{9 \cdot 2^{2k+2} sin^2(\frac{120^o}{2^{k+1}})}$$

$$= 3 \cdot 2^{k+1} sin(\frac{120^o}{2^{k+1}})$$

$$= B_k$$

(Bailey, 2019)

$\frac{A_k}{2}$ and $\frac{B_k}{2}$ are the upper and the lower bounds on the value of π, respectively.

2.3 Results

A program in the language Python was used to compute a specified number of iterations. The program returns half of the perimeter of the larger, circumscribed polygon as the upper bound for π, and half of the perimeter of the smaller, inscribed polygon as the lower bound for π. The upper and lower bounds were compared, and the number of decimal places approximated was calculated as the number of digits starting from the decimal place that were the same between the two bounds. The program can be found in Appendix A.

Table 1: The Decimal Places Approximated and Time Taken After Particular Numbers of Iterations of Archimedes' Method

Iterations	Lower Bound (To the first diverging digit)	Upper Bound (To the first diverging digit)	Decimal Places Approximated	Time Taken (seconds, to 3 significant figures)
1	2	3	0	$1.26*10^{-5}$
5	3.141	3.143	2	$2.34*10^{-5}$
10	3.141592	3.141593	5	$2.75*10^{-5}$
25	3.141592653589792	3.141592653589794	14	$1.28*10^{-4}$

3. Using the Maclaurin Series of the Inverse Tangent Function

3.1 Explanation

It is known that $arctan(1) = \frac{\pi}{4}$. Using Maclaurin series, $arctan(x)$ can be expressed as a polynomial and then used to calculate the value of π.

3.2 Derivation and Proof

The derivative of the inverse tangent function is $\frac{1}{1+x^2}$.

Note the successive derivatives of the inverse tangent function are

$arctan(0) = 0$

$arctan^{(1)}(x) = (1 + x^2)^{-1}$

$\Rightarrow arctan^{(1)}(0) = 1$

$arctan^{(2)}(x) =- 2x(1 + x^2)^{-2}$

$\Rightarrow arctan^{(2)}(0) = 0$

$arctan^{(3)}(x) = (6x^2 - 2)(1 + x^2)^{-3}$

$\Rightarrow arctan^{(3)}(0) =- 2$

$arctan^{(4)}(x) = (- 24x^3 + 24x)(1 + x^2)^{-4}$

$\Rightarrow arctan^{(4)}(0) = 0$

$arctan^{(5)}(x) = (120x^4 - 240x^2 + 24x)(1 + x^2)^{-5}$

$\Rightarrow arctan^{(5)}(0) = 24$

This expression seems impossible to practically continue, but there is a pattern that the nth derivative has value $(-\ 1)^{\frac{n-1}{2}}(n\ -\ 1)!$ at x = 0 if n is odd and zero if n is even, suggesting that the Maclaurin series is:

$$\sum_{n=1}^{\infty}(-\ 1)^{n+1}\frac{(2n-2)!}{(2n-1)!}x^{2n-1}$$

$$=\sum_{n=1}^{\infty}(-\ 1)^{n+1}\frac{1}{(2n-1)}x^{2n-1}$$

The conjecture can be proven within a particular domain using the first derivative of arctan(x). The first derivative of $arctan(x)$ is $\frac{1}{1+x^2}$. Note that the expression for the sum of an infinite series is $\frac{a_n}{1-r}$ where r is the common ratio. Therefore, the inverse tangent function is also an infinite series with first term 1 and common ratio $-\ x^2$. The sum of an infinite series expression only holds if the absolute value of the common ratio is less than 1, and $-\ 1\leq x^2\leq 1$ if and only if $-\ 1\leq x\leq 1$ for real values of x. Thus $\frac{1}{1+x^2}$ can be expressed as the infinite series $1,-\ x^2,x^4,-\ x^6,\ ...$ when $-\ 1\leq x\leq 1$. This domain will be sufficient for the applications within this exploration.

The infinite series can be written in the form

$$1\ -x^2+x^4-x^6+x^8+...=arctan^{(1)}x$$

$$\sum_{n=1}^{\infty}(-\ 1)^{n+1}x^{2n-2}=\ arctan^{(1)}x$$

where $arctan^{(1)}x$ is the first derivative of $arctan(x)$

Both sides can now be integrated

$$\int \sum_{n=1}^{\infty} (-1)^{n+1} x^{2n-2}\, dx = \int arctan^{(1)} x\, dx$$

$$\sum_{n=1}^{\infty} (-1)^{n+1} \int x^{2n-2}\, dx = \int arctan^{(1)} x\, dx$$

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{1}{2n-1} x^{2n-1} = arctan x + C$$

To solve for the constant of integration, set x = 0

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{1}{2n-1} (0)^{2n-1} = arctan(0) + C$$

$$\sum_{n=1}^{\infty} 0 = 0 + C$$

$$0 = C$$

So,

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{1}{2n-1} x^{2n-1} = arctan x$$

Since $arctan(1) = \frac{\pi}{4}$,

$$\sum_{n=1}^{\infty} (-1)^{n+1} \frac{1}{2n-1} (1)^{2n-1} = \frac{\pi}{4}$$

$$\pi = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{4}{2n-1}$$

(Haese, Humphries, Sangwin, & Vo, 2019)

3.3 Results

A program in the language Python was used to compute a specified number of iterations, where each iteration means adding the next pair of positive and negative terms in the Maclaurin series. The value before subtracting the final negative term is returned as the upper bound, while the value after subtracting the final negative term is returned as the lower bound. The upper and lower bounds were compared, and the number of decimal places approximated was calculated as the number of digits starting from the decimal place that were the same between the two bounds. The program can be found in Appendix B.

Table 2: The Decimal Places Approximated and Time Taken After Particular Numbers of Iterations of the Maclaurin Series of the Inverse Tangent Method

Iterations	Lower Bound (To the first diverging digit)	Upper Bound (To the first diverging digit)	Decimal Places Approximated	Time Taken (seconds, to 3 significant figures)
1	2	4	0	$4.39*10^{-5}$
5	3.0	3.3	0	$1.78*10^{-4}$
10	3.1	3.2	0	$3.87*10^{-4}$
25	3.12	3.16	1	$6.20*10^{-4}$

4. Machin’s Formula

4.1 Explanation

The previous method is inefficient because the Maclaurin series of $arctan(x)$ does not quickly converge. However, Machin’s Formula, another formula for approximating π that uses the Maclaurin series for $arctan(x)$, converges much more quickly. Machin’s Formula is:

$$\pi = 16arctan(\frac{1}{5}) - 4arctan(\frac{1}{239})$$

4.2 Derivation and Proof

Let $tan(x) = \frac{1}{5}$

$$tan(2x) = \frac{2tan(x)}{1-tan^2(x)} = \frac{2(\frac{1}{5})}{1-(\frac{1}{5})^2} = \frac{\frac{2}{5}}{1-\frac{1}{25}} = \frac{\frac{2}{5}}{\frac{24}{25}} = \frac{10}{24} = \frac{5}{12}$$

$$tan(4x) = \frac{2tan(2x)}{1-tan^2(2x)} = \frac{2(\frac{5}{12})}{1-(\frac{5}{12})^2} = \frac{\frac{5}{6}}{1-\frac{25}{144}} = \frac{\frac{5}{6}}{\frac{119}{144}} = \frac{120}{119}$$

$$tan(4x - \frac{\pi}{4}) = \frac{tan(4x)-tan(\frac{\pi}{4})}{1+tan(4x)tan(\frac{\pi}{4})} = \frac{\frac{120}{119}-1}{1+\frac{120}{119}*1} = \frac{\frac{1}{119}}{\frac{239}{119}} = \frac{1}{239}$$

$$4x - \frac{\pi}{4} = arctan(\frac{1}{239})$$

$$4arctan(\frac{1}{5}) - \frac{\pi}{4} = arctan(\frac{1}{239})$$

$$4arctan(\frac{1}{5}) - arctan(\frac{1}{239}) = \frac{\pi}{4}$$

$$\pi = 16arctan(\frac{1}{5}) - 4arctan(\frac{1}{239})$$

(Nishiyama, n.d.)

4.3 Results

For both $arctan(\frac{1}{5})$ and $arctan(\frac{1}{239})$, a program in the language Python was used to compute a specified number of iterations, where each iteration means adding the next pair of positive and negative terms in the Maclaurin series.. For both $arctan(\frac{1}{5})$ and $arctan(\frac{1}{239})$, the value before the final negative term is subtracted is returned as the upper bound, while the value after it is subtracted is returned as the lower bound. For the upper bound of π, the upper bound of $arctan(\frac{1}{5})$ and the lower bound of $arctan(\frac{1}{239})$ were used to find the largest possible value. For the lower bound of π, the lower bound of $arctan(\frac{1}{5})$ and the upper bound of $arctan(\frac{1}{239})$ were used to find the smallest possible value. The upper and lower bounds were compared, and the number of decimal places approximated was calculated as the number of digits starting from the decimal place that were the same between the two bounds. The program can be found in Appendix C.

Table 3: The Decimal Places Approximated and Time Taken After Particular Numbers of Iterations of Machin's Method

Iterations	Lower Bound (To the first diverging digit)	Upper Bound (To the first diverging digit)	Decimal Places Approximated	Time Taken (seconds, to 3 significant figures)
1	3.140	3.142	2	$3.50*10^{-5}$
5	3.141592653589791	3.141592653589793	14	$5.51*10^{-5}$
10	31415926535897932 38462643383270	3.1415926535897932 38462643383279	29	$5.67*10^{-5}$
25	3.1415926535897932 38462643383279502 88419716939937510 58209749445923078 16402	3.1415926535897932 38462643383279502 88419716939937510 58209749445923078 16406	71	$7.70*10^{-5}$

5. Analysis and Conclusion

5.1 Comparison of the Methods

The value of π to some particular decimal places was calculated using both Archimedes' method and Machin's method, and the iterations required and the time taken in seconds was recorded. The Maclaurin series of the inverse tangent method was not used because it is apparent that it is too inefficient. 15 was chosen because it is the number of decimal places used by NASA's Jet Propulsion Laboratory in their most precise calculations, while 37 was used as it is the number of decimal places needed the calculate the circumference of the observable universe to a precision equal to the diameter of a hydrogen atom (Rayman, 2022). 100 and 1000 were chosen as large numbers to extrapolate the relative speeds of the methods as the number of decimal places increases.

Table 3: The Iterations and Time Necessary to Approximate π to Various Decimal Places Using Archimedes' Method and Machin's Method

Decimal Places Approximated	Archimedes' Method		Machin's Method	
	Iterations	Time Taken (Seconds)	Iterations	Time Taken (Seconds)
15	26	$1.17*10^{-4}$	6	$1.69*10^{-5}$
37	65	$1.26*10^{-3}$	13	$2.50*10^{-5}$
100	168	$1.07*10^{-2}$	36	$6.68*10^{-5}$
1000	1662	6.55	357	$2.15*10^{-3}$

5.2 Real World Applications

As circles appear ubiquitously in practical applications, approximating π to some precision is necessary for a variety of tasks from designing buildings to launching spacecraft. However, as each decimal place increases the precision by an order of magnitude, there is a point at which other sources of error mask uncertainty from not having the precise value of π. With enough decimal places of π approximated, the difference from the exact value becomes irrelevant when that level of precision cannot be achieved in the real world or is not necessary for the task. According to NASA's Jet Propulsion Laboratory, just 37 decimal places of π can be used to calculate the circumference of a circle with the same radius as the observable universe to a precision of the diameter of one hydrogen atom. As a result, the Jet Propulsion Laboratory uses at most 15 decimal places of π for its calculations. Even fewer decimal places are required for the vast majority of real world applications, and there is little practical use to knowing the value of π to more precision than this (Rayman, 2022).

However, even if knowing the value of π to trillions of decimal places is not necessary, approximating it can still be useful. Approximating π is often used as a method to test the computational power of both hardware and software, as methods have become more refined over time but inevitably require a huge number of calculations. Currently, the world record for most digits of π approximated is 100 trillion. This approximation was done by Google using the Chudnovsky Algorithm, and took 157 days, 23 hours, 31 minutes, and 7.651 seconds (Iwao, 2022).

$$\frac{\sqrt{640320^3}}{12\pi} = \sum_{n=0}^{\infty} \frac{(6n)!}{(3n)!\,(n!)^3} \frac{13591409 + 545140134n}{\left(-640320^3\right)^n}$$

Figure 5: Chudnovsky Algorithm (Milla, 2021)

5.3 Limitations and Improvements

One limitation in this exploration was the hardware and software used. The programs for approximating π were run on the language Python, known to be slower and less efficient than other programming languages such as C++. Using another language could potentially improve computational speed. Additionally, the programs were executed on a personal computer. Using a dedicated processor might have also improved efficiency. This exploration was limited to identifying means of approximating π and compare their relative efficiencies, but improved hardware and software could allow for the investigation the computational power needed to approximate π , and how computations could be optimized.

5.4 Areas of Further Exploration

An area for further exploration are proving and evaluating modern methods for approximating π such as the Chudnovsky Algorithm. Another area for future exploration is using computing to approximate other important mathematical constants such as Euler’s number.

5.5 Conclusion

This exploration investigated several methods of approximating π, using Archimedes' method and by using Machin's method. The exploration found that Machin's method could approximate π to several numbers of decimal places faster than Archimedes's method can. This can be attributed to the fact that Machin's method takes fewer iterations to approximate π to the same number of decimal places, meaning that Machin's method converges much faster on π.

This exploration was successful in answering my question of how mathematicians approximate π by giving me an opportunity to compute it in several different ways. Throughout this exploration, I researched different methods that mathematicians used to calculate π, and devised my own Python code to do the computations iteratively. I also realized that the number of decimal places approximated could be determined by comparing upper and lower bounds and counting the number of digits after the decimal that are the same in both. In evaluating the results, the difference in efficiency between Archimedes' method and Machin's method gave me unique insight into how breakthroughs in mathematics, such as the invention of calculus, have allowed for further refining of algorithms such as the ones used to approximate π. Additionally, using a computer to do thousands of calculations in seconds, which would have taken hours by hand, has shown me how computers have revolutionized mathematics.

Works Cited

Bailey, D. H. (2019, February 9). *Simple proofs: Archimedes' calculation of pi*. Math Scholar. Retrieved January 4, 2023, from https://mathscholar.org/2019/02/simple-proofs-archimedes-calculation-of-pi/

Haese, M., Humphries, M., Sangwin, C., & Vo, N. (2019). *Mathematics: Analysis and Approaches Hl*. Oxford University Press.

Iwao, E. H. (2022, June 8). *Even more pi in the sky: Calculating 100 trillion digits of pi on Google Cloud*. Google. Retrieved February 20, 2023, from https://cloud.google.com/blog/products/compute/calculating-100-trillion-digits-of-pi-on-google-cloud

Milla, L. (2021). An efficient determination of the coefficients in the Chudnovskys' series for 1/π. *The Ramanujan Journal*, *57*(2), 803–809. https://doi.org/10.1007/s11139-020-00330-6

Nishiyama, Y. (n.d.). *Machin's Formula and Pi*. Bugianens' Maccaja. Retrieved February 16, 2023, from http://personalpages.to.infn.it/~zaninett/pdf/machin.pdf

Rayman, M. (2022, November 30). *How Many Decimals of Pi Do We Really Need?* NASA. Retrieved February 20, 2023, from https://www.jpl.nasa.gov/edu/news/2016/3/16/how-many-decimals-of-pi-do-we-really-need/

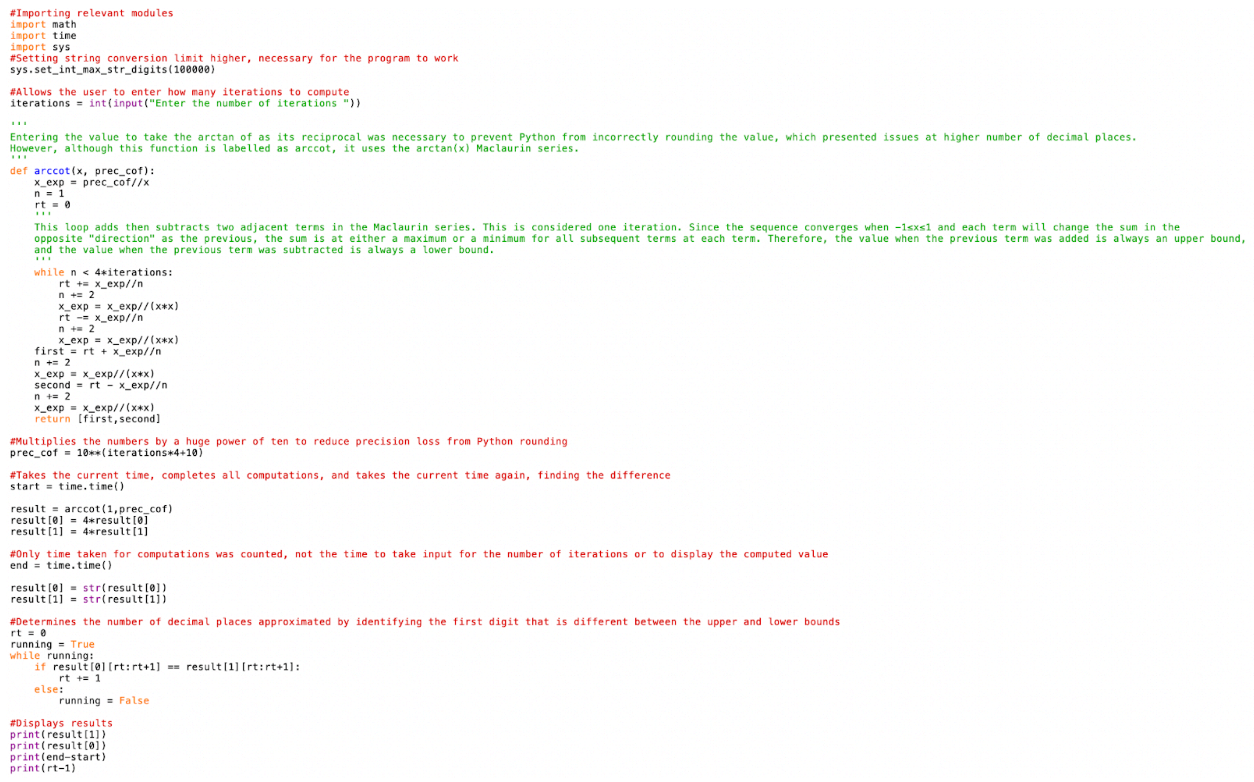
Appendix A: Program for Archimedes' Method

This program returns half of the perimeter of the larger, circumscribed polygon as the upper bound for π, and half of the perimeter of the smaller, inscribed polygon as the lower bound for π. To prevent precision loss from rounding, the values were multipled by a large power of ten while being calculated and displayed. It should be known by users of this program that the decimal is meant to be placed immediately after the first digit.

```python
#Importing relevant modules
import math
import time
import sys
sys.set_int_max_str_digits(100000)
#Setting string conversion limit higher, necessary for the program to work
sys.set_int_max_str_digits(100000)

#Allows the user to enter how many iterations to compute
iterations = int(input("Enter the number of iterations "))

#Multiplies the numbers by a huge power of ten to reduce precision loss from Python rounding
prec_con = 10**(iterations*4+10)

#Iteratively computes the perimeters of the inscribed and circumscribed polygons
def polygon_method(iterations):
    greater_perimeter = 6*math.isqrt(3*(prec_con)**2)
    lesser_perimeter = 3*math.isqrt(3*(prec_con)**2)
    i = 0
    while i < iterations:
        greater_perimeter = (2*greater_perimeter*lesser_perimeter)//(greater_perimeter+lesser_perimeter)
        lesser_perimeter = math.isqrt(greater_perimeter*lesser_perimeter)
        i=i+1
    return [lesser_perimeter//2, greater_perimeter//2]

#Takes the current time, completes all computations, and takes the current time again, finding the difference
start = time.time()

result = polygon_method(iterations)

#Only time taken for computations was counted, not the time to take input for the number of iterations or to display the computed value
end = time.time()

result[0] = str(result[0])
result[1] = str(result[1])

#Determines the number of decimal places approximated by identifying the first digit that is different between the upper and lower bounds
rt = 0
running = True
while running:
    if result[0][rt:rt+1] == result[1][rt:rt+1]:
        rt += 1
    else:
        running = False

#Displays results
print(result[1])
print(result[0])
print(end-start)
print(rt-1)
```

Appendix B: Program for Maclaurin Series of the Inverse Tangent Function Method

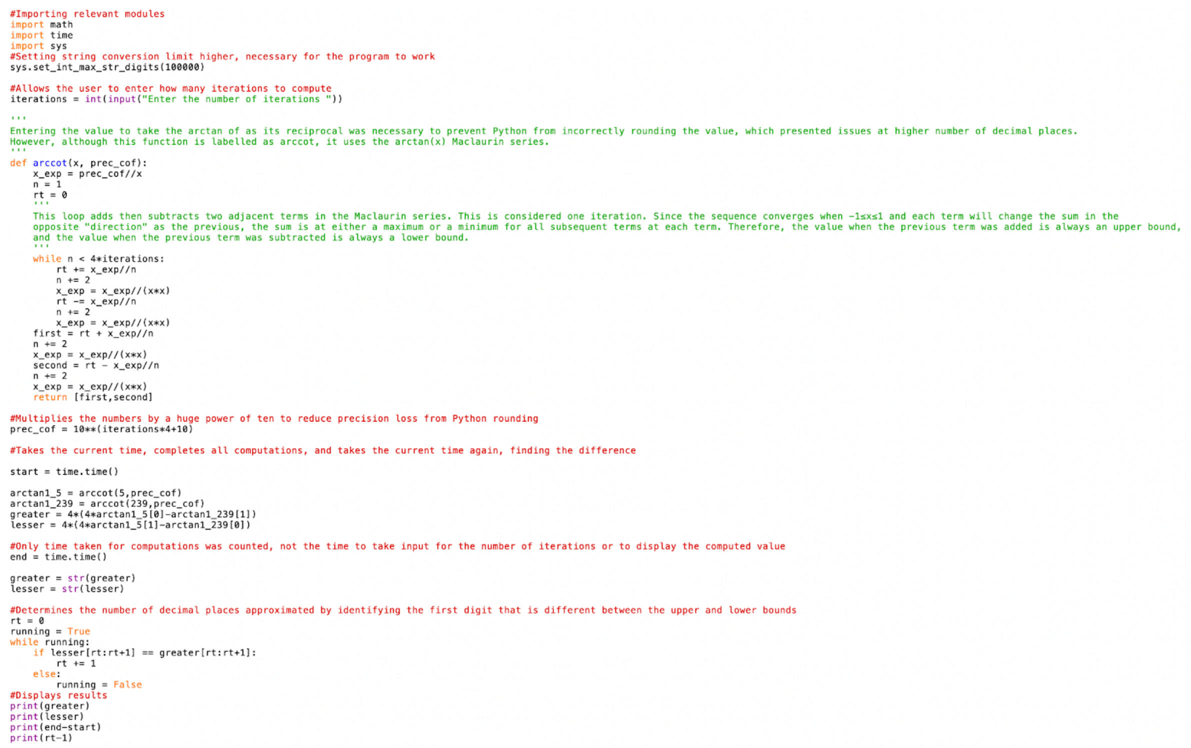
This program computes a number of pairs of positive and negative terms equal to the specified number of iterations. The value before the final negative term is subtracted is returned as the upper bound, while the value after it is subtracted is returned as the lower bound. To prevent precision loss from rounding, the values were multipled by a large power of ten while being calculated and displayed. It should be known by users of this program that the decimal is meant to be placed immediately after the first digit.



```python
#Importing relevant modules
import math
import time
import sys
#Setting string conversion limit higher, necessary for the program to work
sys.set_int_max_str_digits(100000)

#Allows the user to enter how many iterations to compute
iterations = int(input("Enter the number of iterations "))

'''
Entering the value to take the arctan of as its reciprocal was necessary to prevent Python from incorrectly rounding the value, which presented issues at higher number of decimal places.
However, although this function is labelled as arccot, it uses the arctan(x) Maclaurin series.
'''
def arccot(x, prec_cof):
    x_exp = prec_cof//x
    n = 1
    rt = 0
    '''
    This loop adds then subtracts two adjacent terms in the Maclaurin series. This is considered one iteration. Since the sequence converges when -1≤x≤1 and each term will change the sum in the
    opposite "direction" as the previous, the sum is at either a maximum or a minimum for all subsequent terms at each term. Therefore, the value when the previous term was added is always an upper bound,
    and the value when the previous term was subtracted is always a lower bound.
    '''
    while n < 4*iterations:
        rt += x_exp//n
        n += 2
        x_exp = x_exp//(x*x)
        rt -= x_exp//n
        n += 2
        x_exp = x_exp//(x*x)
    first = rt + x_exp//n
    n += 2
    x_exp = x_exp//(x*x)
    second = rt - x_exp//n
    n += 2
    x_exp = x_exp//(x*x)
    return [first,second]

#Multiplies the numbers by a huge power of ten to reduce precision loss from Python rounding
prec_cof = 10**(iterations*4+10)

#Takes the current time, completes all computations, and takes the current time again, finding the difference
start = time.time()

result = arccot(1,prec_cof)
result[0] = 4*result[0]
result[1] = 4*result[1]

#Only time taken for computations was counted, not the time to take input for the number of iterations or to display the computed value
end = time.time()

result[0] = str(result[0])
result[1] = str(result[1])

#Determines the number of decimal places approximated by identifying the first digit that is different between the upper and lower bounds
rt = 0
running = True
while running:
    if result[0][rt:rt+1] == result[1][rt:rt+1]:
        rt += 1
    else:
        running = False

#Displays results
print(result[1])
print(result[0])
print(end-start)
print(rt-1)
```

Appendix C: Program for Machin's Method

This program computes a number of pairs of positive and negative terms equal to the specified number of iterations. For both $arctan(\frac{1}{5})$ and $arctan(\frac{1}{239})$, the value before the final negative term is subtracted is returned as the upper bound, while the value after it is subtracted is returned as the lower bound. For the upper bound of π, the upper bound of $arctan(\frac{1}{5})$ and the lower bound of $arctan(\frac{1}{239})$ were used to find the largest possible value. For the lower bound of π, the lower bound of $arctan(\frac{1}{5})$ and the upper bound of $arctan(\frac{1}{239})$ were used to find the smallest possible value. To prevent precision loss from rounding, the values were multipled by a large power of ten while being calculated and displayed. It should be known by users of this program that the decimal is meant to be placed immediately after the first digit.



```python
#Importing relevant modules
import math
import time
import sys
#Setting string conversion limit higher, necessary for the program to work
sys.set_int_max_str_digits(100000)

#Allows the user to enter how many iterations to compute
iterations = int(input("Enter the number of iterations "))

'''
Entering the value to take the arctan of as its reciprocal was necessary to prevent Python from incorrectly rounding the value, which presented issues at higher number of decimal places.
However, although this function is labelled as arccot, it uses the arctan(x) Maclaurin series.
'''
def arccot(x, prec_cof):
    x_exp = prec_cof//x
    n = 1
    rt = 0
    '''
    This loop adds then subtracts two adjacent terms in the Maclaurin series. This is considered one iteration. Since the sequence converges when -1≤x≤1 and each term will change the sum in the
    opposite "direction" as the previous, the sum is at either a maximum or a minimum for all subsequent terms at each term. Therefore, the value when the previous term was added is always an upper bound,
    and the value when the previous term was subtracted is always a lower bound.
    '''
    while n < 4*iterations:
        rt += x_exp//n
        n += 2
        x_exp = x_exp//(x*x)
        rt -= x_exp//n
        n += 2
        x_exp = x_exp//(x*x)
    first = rt + x_exp//n
    n += 2
    x_exp = x_exp//(x*x)
    second = rt - x_exp//n
    n += 2
    x_exp = x_exp//(x*x)
    return [first,second]

#Multiplies the numbers by a huge power of ten to reduce precision loss from Python rounding
prec_cof = 10**(iterations*4+10)

#Takes the current time, completes all computations, and takes the current time again, finding the difference

start = time.time()

arctan1_5 = arccot(5,prec_cof)
arctan1_239 = arccot(239,prec_cof)
greater = 4*(4*arctan1_5[0]-arctan1_239[1])
lesser = 4*(4*arctan1_5[1]-arctan1_239[0])

#Only time taken for computations was counted, not the time to take input for the number of iterations or to display the computed value
end = time.time()

greater = str(greater)
lesser = str(lesser)

#Determines the number of decimal places approximated by identifying the first digit that is different between the upper and lower bounds
rt = 0
running = True
while running:
    if lesser[rt:rt+1] == greater[rt:rt+1]:
        rt += 1
    else:
        running = False
#Displays results
print(greater)
print(lesser)
print(end-start)
print(rt-1)
```